

# **DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY**

P. Little

*Prepared for*  
**NASA/JSC Data Management Systems  
for Space Station Freedom (SSF)**

Prepared by Co-Principal Investigators:

**T. F. Leibfried Jr.  
Sadegh Davari**  
*University of Houston-Clear Lake*

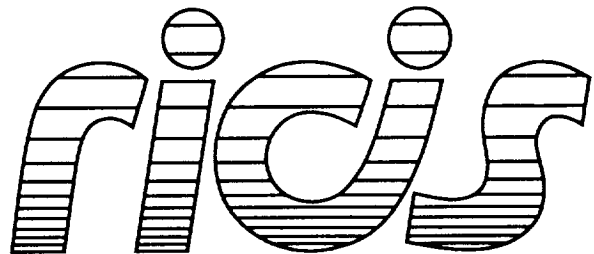
**Swami Natarajan  
Wei Zhao**  
*Texas A&M University*

Research Associates:

**Libin Wu**  
*University of Houston-Clear Lake*

**Gary Smith**  
*Texas A & M University*

April 1992



*Research Institute for Computing and Information Systems  
University of Houston-Clear Lake*

(NASA-CR-190397) DATA MANAGEMENT SYSTEMS  
(DMS): COMPLEX DATA TYPES STUDY. VOLUME 1:  
APPENDICES A-B. VOLUME 2: APPENDICES C1-C5.  
VOLUME 3: APPENDICES D1-D3 AND E Final  
Report (Research Inst. for Computing and

N92-26552

71819 Unclass  
G3/81 0096739

## ***The RICIS Concept***

---

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

*Prepared for*

***NASA/JSC Data Management Systems  
for Space Station Freedom (SSF)***





## **Preface**

This research was conducted under the auspices of the Research Institute for Computing and Information Systems. Co-Principal Investigators for this activity were T.F. Leibfried, Jr. and Sadegh Davari of the University of Houston-Clear Lake and Swami Natarajan and Wei Zhao of Texas A&M University. They were aided by research assistants Libin Wu from UHCL and Gary Smith of TAMU. Colin Atkinson of UHCL helped formulate and structure the technical diagrams and their contents.

Funding was provided by RICIS Research Support through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The equipment and software used was purchased under a grant from the Performance and Integration Branch of the Communications and Tracking Division at NASA/JSC. Some of the compiler and operating system software was supplied by the Alsys Corporation and the Lynx Corporation.

We would also like to acknowledge the assistance of Mr. Ken Westerfeld, the DMS Systems Software Section Head, Mr. William Dwyer, the DMS Software Development Manager and Ms. Diana Barber, the STSV Software Specialist, all from the same NASA/JSC Flight Systems Division, for their helpful comments and documentary information. In addition, the helpful suggestions of Dr. Robert Brown of Draper Laboratories are acknowledged.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of UHCL, RICIS, NASA or the United States Government.



# Table of Contents

Summary and Executive Overview	1
Introduction	1
Summary of Investigations and Results	2
Preprocessor Study	2
RODB STSV Performance Analysis Study	2
Summary of Conclusions	5
Final Report of Data Management systems Complex Data Types Study	6
Introduction	6
RODB User Interface (Preprocessor Description)	8
Part I: The Current Interface	9
Part II: Issues	11
Potential Problems with the User Interface	11
Part III: Solutions	13
Part IV: Comparison of Design 1 and Design 2	15
Performance Analysis	16
1. IPC Message Queue Performance	16
2. Shared Memory Access Performance	17
3. Test of Mutual Exclusion for the Reader-Writer Problem	22
Summary of Observations and Suggestions	25
References	26
Abbreviated Appendices	27



# Summary and Executive Overview

## Introduction

The Complex Data Types problem study, a voluntary RICIS study inaugurated with the advice of the Data Management Systems Branch of the NASA Johnson Space Center (JSC) Flight Data Systems Division began during the summer of 1990. Of the important prioritized issues two categories were chosen for investigation. These were:

1. The issue of using a preprocessor on Ada code of Application Programs which would interface with the Run-Time Object Data Base Standard Services (RODB STSV); The intent was to catch and correct any mis-registration errors of the program coder between the user declared Objects, their types, their addresses and the corresponding RODB definitions;
2. RODB STSV Performance Issues and Identification of Problems with the planned methods for accessing Primitive Object Attributes. This included the investigation of an alternate storage scheme to the store-objects-by-attribute scheme in the current design of the RODB.

These two efforts were pursued in parallel the preprocessor effort produced a preliminary prototype in the summer of 1991. With respect to the RODB and STSV the study resulted in essentially three separate documents, an interpretation of the system requirements, an assessment of the preliminary design and a detailing of the components of a detailed design. These documents were produced and delivered in the Spring and Summer of 1991. The requirements document was produced in early April [10], the preliminary design document was produced in late April [9] and the detailed design document detailing the potential issues and problems was produced in July of 1991 [8]. The code for the preprocessor was delivered in August of 1991.



# Summary of Investigations and Results

## Preprocessor Study

The preprocessor was designed to perform a simplified Lexical analysis of an application's Ada code. Specifically it was intended to review that portion of the code which interfaced to the Application Program Interface of the Standard Services. In particular the part of the application's code which was of greatest interest was that which contained the definitions of the so-called object-attribute arrays and the corresponding attribute address arrays. The preprocessor was structured to receive Ada code which had successfully compiled and so was free of syntactical errors. The output of the preprocessor is modified application code (i.e. additional object and attribute definitions). This code, when again subjected to the Ada compiler reveals some of the semantic errors in the original code.

### *Preprocessor Study Results*

The fundamental results of the experiments ran indicated what was suspected almost from the beginning. The preprocessor could easily detect when there was a mis-registration of two attribute elements which were of different primitive types. It could not detect when there was a mis-registration between two attributes of the same primitive type. The preliminary conclusion is that with the present design, in order to insure software that is essentially free of semantic errors, it will be necessary to severely restrict the range of names that an application programmer may use in writing code.

## RODB STSV Performance Analysis Study

The study began with an intensive review of the RODB and STSV documents. The two most important of which were the Detailed Design Document and the STSV Interface Requirements Document (IRD). In the IRD specifically the Application Program Interface Definition (APIID) was of greatest interest. Both of these documents were IBM Documents which had been submitted to the prime contractor McDonnell-Douglas.

As mentioned in the introduction there were three intermediate reports generated, viz. Prototype Requirements, Prototype Preliminary Design and Detailed Design documents. From these it was determined that there were two principal areas where performance bottlenecks or problem areas might appear. All of these areas may be classified under the broad general heading of Inter-Process

Communication (IPC). In fact all three forms of IPC as classified by the UNIX operating system are involved. Specifically so-called IPC Message Queues, Semaphores and Shared memory.

The two areas of concern are the communications between the Application (API) and the Application Interface Service (AIS) and the accessing, (i.e. reading and writing to), the RODB Component itself. The AIP-AIS communication path consists of two sets of message queues; one is the application request IPC Message Queue, the other being the response IPC Message Queues. The RODB Component "communications" consists of shared memory with "guarding" semaphores. There were other areas of concern such as inter-task communication (ITC) but these are wholly within the Ada environment and were deemed to be areas where study could be postponed until funds were available.

The IPC Message Queues were originally planned for use between Application Programs and the RODB Executive system (REX) to open various types of reads and writes to the RODB Component as well as to confirm the validity of the handles during an actual read or write. The shared memory and the semaphores were and are to be used during the actual reading and writing of data.

The open operation for the RODB is a STSV function which by its nature must use an IPC of some sort and the chosen facility is that of IPC Message Queues. What essentially transpires is for the application to forward a request to the STSV to allow the reading or writing of a set of Object-Attribute pairs. After checking permissions etc. STSV constructs what is called a "handle", which is a set of addresses of the appropriate shared memory segment where the pertinent attributes are stored. This information is then passed back through a response queue to the requesting application. If nothing causes this information to change the open operation need only be done once. Multiple reads may take place as long as the handle is not closed. The handle is a structure maintained by REX to provide validity and access information to the RODB Component objects.

The read or write operation originally required an additional request through the IPC Message Queue mechanism to verify the validity of the handle(s). Once a favorable acknowledgment was received through a response message queue, the application was then free to access the appropriate shared memory addresses to read or write the actual variable data. At that point there was to be a relatively classic Readers-Writers problem in that multiple readers were to be permitted to access the same object attribute, but when readers were reading writers must wait. Further when a writer is writing readers must wait until the writer finishes. While the actual mechanism of the reader-writer was not detailed in the documents the aforementioned policy dictated at least one set of possible semaphores to solve the reader-writer problem for the RODB Component shared memory.

## ***Performance Tests Implementation and Results***

When the Lynx Operating System and the Alsys Ada Compiler were received several test code segments were constructed and run. The platform used to approximate the Embedded Data Processor (EDP) to be used on the Flight Data Systems for the SSF was a PS/2 Model 80 with a 20 MHz clock rate



The first Complex Data Types test run on this platform was the IPC Message Queue implementation of the Request/Response message queues. A simple message was sent between two C programs with an acknowledgment message sent in return. If this was necessary for each write and/or read, it was postulated that for post "Scrub," with only two application EDP's on the SSF and an estimated 10,000 sensors, that 5000 writes might be required for each EDP each cycle. The original intent was for the cycle time to be every second. When the test was run, it was found that 5000 IPC Message round trips could be accomplished in no less than approximately six (6) seconds. Thus it appeared that the use of IPC Message Queues for every read or write would not be feasible if the plan was even to come close to the original requirement. A similar result had been found on a Harris HCX-9, which is supposedly a much faster machine. This was duly reported to the NASA DMS personnel. It is interesting to note that IBM either suspected this fact or independently determined this since the latest design does not require IPC Message Queues for every read or write but only for the open and close operations. Instead of requiring validation of the handles for every read or write there is a RODB Component version number returned as part of the handle from the open call. A corresponding number is stored in the RODB Component itself so the validation check is done when the actual read or write is performed.

The other subsystem potential problem area investigated was the accessing of the RODB Shared Memory itself. Here both a Shared Memory component and IPC Semaphores were implemented. Again 5000 noncompeting reads or writes per cycle were postulated and simple begin and end times were recorded. In this case if 5000 reads or writes were performed a total number of between two (2) and three (3) seconds were required. In addition a suspected deficiency of the semaphore operation of the Lynx Version 2.0 for the 386 microprocessor was found. It was found that when the Readers-Writers problem was implemented the semaphore semop system call operation did not perform in a manner consistent with AT&T UNIX System V semaphores. In fact atomicity of an array-of-semaphore operations was not enforced so that in the Readers-Writers implementation a reader and a writer could be accessing the same RODB Component at the same time. When priority was raised to work around this problem, and the semop system call was repeated, the performance deteriorated to requiring over seven (7) seconds to perform 5000 reads or writes. When concurrent and competing read and write processes were tested, the time for two readers doing 2500 reads each and a writer writing 5000 time to the same attributes took a maximum of 19 seconds to complete. When the code for dynamically changing the priorities was removed and just the semaphore operations were left in tact that time dropped to seven (7) seconds. Then, however, mutual exclusion was not assured. It is interesting although probably moot to note that if 5000 unprotected concurrent reads and writes are performed, the elapsed time is about two (2) seconds. What seems indicated here is that semaphore operations may require some lower level coding to achieve desired or near-desired results.

Detailed results and test programs are included in the body and appendices of this report respectively.

## Summary of Conclusions

It seems clear that research and advanced development investigations have the beneficial effect of identifying problem areas for even proposed production projects where the state of the art is being “pushed”. The problems identified are:

1. In the absence of an advanced Object Oriented interface the naming conventions for the application objects and attributes will have to be severely restricted.
2. The use of IPC mechanisms, though necessary for communication between different processes, adds cost to the processing time. It may be necessary to investigate modifying the low level code of the operating system to specialize the system calls for efficiency. In particular the dynamic changing of priorities and IPC Semaphore operations are costly.
3. There is another issue on the semaphores which must be addressed at some point. It appears that the system call semop does not fully meet the execution of a system call. This is serious since, with this deficiency, there is no guarantee that the semaphores guarding the shared memory will ensure -- in the implementation tested-- that several processes will not be in their critical sections at the same time.

One alternative to storing RODB data by attribute would be to store by object. While this concept is closer to what is normally done by many operating systems for user text and data. In this study trade-off studies were only superficially addressed and quantitative studies of these object concepts could not be done because of limited knowledge of data needs of the applications and limited resources. Similarly, other problem areas such as the problems of dynamic scheduling and dynamic upgrade as well as interoperability problems and the issue of designing more advanced object oriented interfaces, among other issues, remain unsolved.

# Final Report of Data Management Systems Complex Data Types Study

## Introduction

The Space Station Freedom (SSF) Data Management System (DMS) includes a distributed but centrally controlled data base and concomitant services. The Run-Time Object Data Base (RODB) and Standard Services (STSV) structure for the Space Station is intended to provide data acquisition and preliminary processing for all applications on the SSF.

The RODB provides a current-time data repository which is the medium through which the various applications communicate with each other and under Mission Control.

The RODB, for simplicity, only provides the capability of storing primitive data types such as integers, floats, enumeration and pointer types. The exception to this limitation is the capability of storing linear arrays of these primitive types. Records and matrices were considered complex types and as such were not admissible but had to be broken down into their component fields before entering them into the RODB. Since records could not be written or read as such from the RODB there were consistency and performance issues which deserved investigation; hence this study was inaugurated as a sample of relevance of academic research and development to "real-world" problems by the UHCL Research Institute for the Computing and Information Sciences (RICIS), with the cooperation of the UHCL and TAMU Computer Science Faculties.

Two salient areas seemed to be evident when the functional design was chosen to meet the basic design represented by "SOFTWARE PRELIMINARY DESIGN DOCUMENT (DATA MANAGEMENT SYSTEM STANDARD SERVICES) "[1],.

The detailed design document entitled "SOFTWARE DETAILED DESIGN DOCUMENT (DATA MANAGEMENT SYSTEM STANDARD SERVICES) "[2], contained the information which outlined essentially the current structure of the RODB and the STSV. These two broad areas of concern became evident when Dr. Swami Natarajan of TAMU studied the requirements and preliminary design for the DMS, RODB and SSTV.

One of these problem areas was the interfaces between the applications and the DMS. The applications software units were, by nature, separate executable programs. Any communication between them and the DMS service processes would have to use the inter-process communication mechanisms of the underlying operating system. The DMS architecture supplies packages for the applica-

# DMS STSV RODB Flow Control

As described in DMS Det. Des.

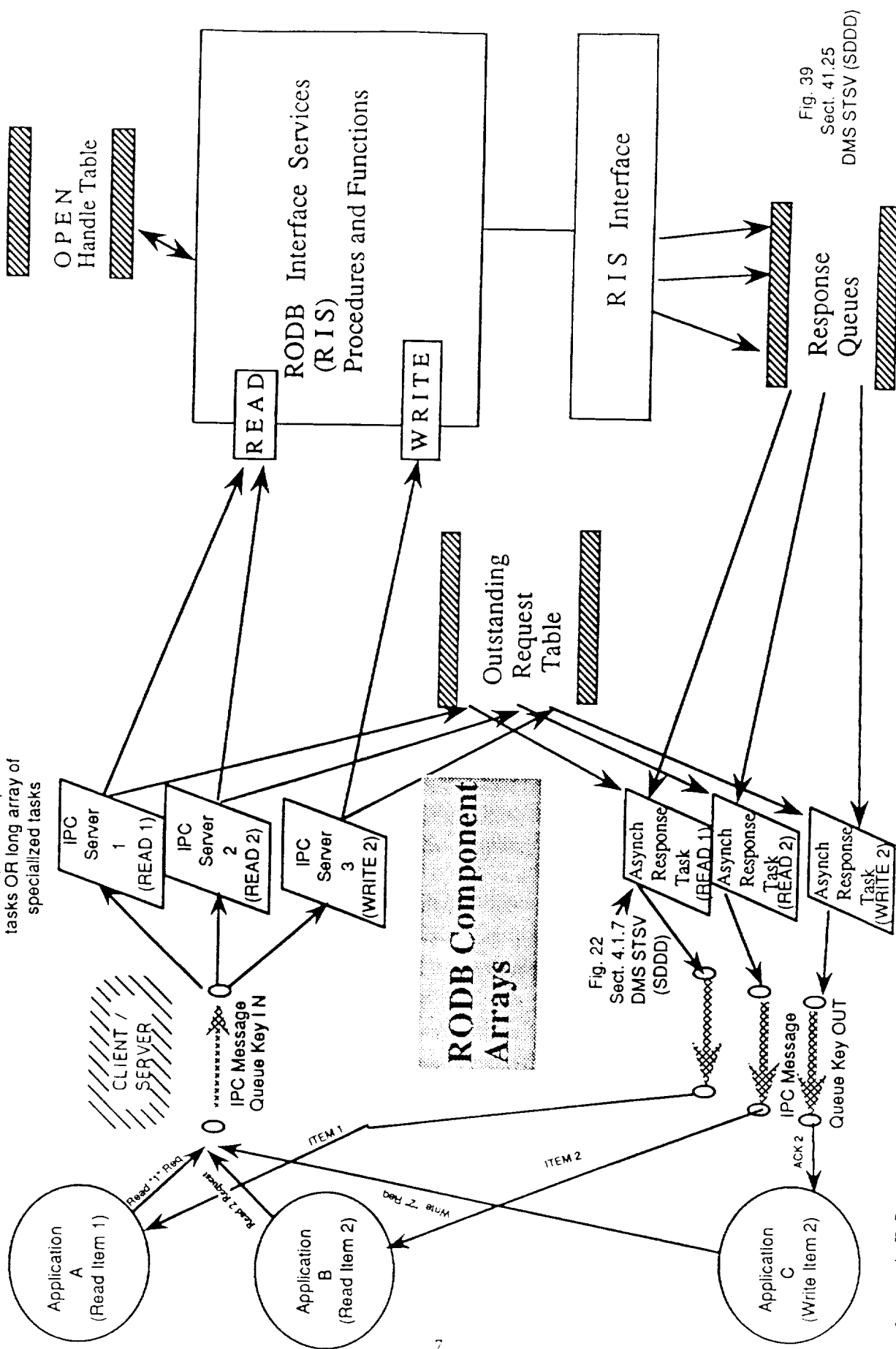


Fig. 39  
Sect. 41.25  
DMS STSV (SDDD)

Figure 1: RODB IPC/ITC Message Queue Flow

TFL/LW/CA Feb 92

tions to utilize. These packages are known as the Application Program Interface (API). They are Ada packages which the application programs can "with" and which supply the requisite communication mechanisms. Unfortunately the interface between an application processes and the DMS service processes circumvents the strong typing checks of any Ada compiler. This implies that some preprocessing might mitigate the potential inconsistency problem between the two subsystems (i.e. Application and DMS). The description of this interface is in McDonnell Douglas document No. MDC H4193 APPENDIX I entitled "INTERFACE REQUIREMENTS DOCUMENT (SOFTWARE) APPENDIX I INTERFACE CONTROL DOCUMENT (ICD) DATA MANAGEMENT SYSTEM (IBM)",[3].

The other area of concern could be summed up as the potentially deleterious effect of the complexity of inter-process communication upon the performance or throughput of the system. Both of these concerns are illustrated on Figure 1. "RODB IPC/ITC Message Queue Flow". The portion of the figure labeled API shows, as an example, three application processes which are linked to the DMS via the API packages. The rest of the figure illustrates the mechanisms for inter-process communication (IPC) and inter-task communication (ITC).

## **RODB User Interface (Preprocessor Description)**

### **Problem Statement**

Applications use the RODB to communicate with each other and also to obtain data from sensors and send data to effectors. The particular problem we addressed in this study was using Attribute I/O for the user interface for reading and writing data to and from the RODB. The potential of type errors in using that scheme and the use of a compile-time preprocessor to catch these errors was investigated.

### **Outline of Preprocessor Discussion**

The purpose here is to examine the current proposed user interface to the RODB, identify any problems which may exist, suggest modifications which may overcome some of these problems, and evaluate the alternative approaches. The discussion in this section is divided into five parts. The first part presents the current interface. The second part identifies some problems with the interface. In part three, we list and discuss some of the approaches which may be used to overcome these problems. In the fourth part, we provide a comparative evaluation of these approaches. In part five, Appendix A, we outline the design of a preprocessor to address some of these problems.

## Part I: The Current Interface

To read or write a record, applications must enumerate the individual fields involved - the interface has no concept of records. The programmer must construct two kinds of lists at compile-time. The first is an attribute list, which specifies the type of each of the fields which will be read (or written). Each entry in the attribute list consists of a pair of values, the first element of the pair being the type of the object whose fields are to be read, and the second element being the type of the particular field which is to be read. Thus a typical attribute list may be:

```
APT_POSITION_ATTRIBUTE_ARRAY: constant STRDB.ATTRIBUTE_ARRAY_T :=  
  ((AP_TELESCOPE, RIGHT_ASC),  
   (AP_TELESCOPE, DEC_DEGREES),  
   (AP_TELESCOPE, DEC_MINUTES),  
   (AP_TELESCOPE, DEC_SECONDS));
```

There is no restriction that all the entries in the list must be fields from the same type of object; indeed, an advantage of the interface is the flexibility it provides in reading and writing fields in different records with a single RODB invocation. The programmer must construct one of these attribute arrays for each different group of field types.

The attribute array is provided as a parameter in the call to open an RODB handle:

```
READ_OPEN (ATTRIBUTE_LIST => APT_POSITION_ATTRIBUTE_ARRAY,  
           HANDLE         => APT_POSITION,  
           HANDLE_ID      => APT_HANDLE_ID,  
           ERRORS_DETECTED => ERRORS_DETECTED,  
           ERRORS         => ERRORS);
```

This call initializes a set of internal RODB handles to access the desired combination of fields, and returns a pointer to this handle to the user.

For each set of data values to be read or written, the programmer must also construct another list consisting of the actual addresses where the data is located. Thus an address list for the above example might be:

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  (APT_POSITION_RECORD.RIGHT_ASCENSION'address,  
   APT_POSITION_RECORD.DECLINATION.DEGREES'address,  
   APT_POSITION_RECORD.DECLINATION.MINUTES'address,  
   APT_POSITION_RECORD.DECLINATION.SECONDS'address);
```

Each element of this address array must match exactly in type the corresponding elements of the attribute array, so that values of the correct types will be placed in the corresponding locations.

The address array is provided as a parameter on the READ call:

```
ATTRIBUTE_READ (HANDLE_LIST => LIST OF (APT_POSITION),  
                POSITIONS   => APT_POSITION_ADDRESS_ARRAY,  
                ERRORS_DETECTED => ERRORS_DETECTED,  
                ERRORS       => ERRORS);
```

```
APT_POSITION_ATTRIBUTE_ARRAY: constant STRDB.ATTRIBUTE_ARRAY_T :=  
  ((AP_TELESCOPE, RIGHT_ASC),  
   (AP_TELESCOPE, DEC_DEGREES),  
   (AP_TELESCOPE, DEC_MINUTES),  
   (AP_TELESCOPE, DEC_SECONDS));
```

There is no restriction that all the entries in the list must be fields from the same type of object; indeed, an advantage of the interface is the flexibility it provides in reading and writing fields in different records with a single RODB invocation. The programmer must construct one of these attribute arrays for each different group of field types.

The attribute array is provided as a parameter in the call to open an RODB handle:

```
READ_OPEN (ATTRIBUTE_LIST => APT_POSITION_ATTRIBUTE_ARRAY,  
           HANDLE         => APT_POSITION,  
           HANDLE_ID      => APT_HANDLE_ID,  
           ERRORS_DETECTED => ERRORS_DETECTED,  
           ERRORS         => ERRORS);
```

This call initializes a set of internal RODB handles to access the desired combination of fields, and returns a pointer to this handle to the user.

For each set of data values to be read or written, the programmer must also construct another list consisting of the actual addresses where the data is located. Thus an address list for the above example might be:

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  (APT_POSITION_RECORD.RIGHT_ASCENSION'address,  
   APT_POSITION_RECORD.DECLINATION.DEGREES'address,  
   APT_POSITION_RECORD.DECLINATION.MINUTES'address,  
   APT_POSITION_RECORD.DECLINATION.SECONDS'address);
```

Each element of this address array must match exactly in type the corresponding elements of the attribute array, so that values of the correct types will be placed in the corresponding locations.

The address array is provided as a parameter on the READ call:

```
ATTRIBUTE_READ (HANDLE_LIST => LIST OF (APT_POSITION),  
                POSITIONS   => APT_POSITION_ADDRESS_ARRAY,  
                ERRORS_DETECTED => ERRORS_DETECTED,  
                ERRORS       => ERRORS);
```

The RODB code accesses its internal data using the handles, and transfers each data item into the corresponding address provided in the address array.

## **Part II: Issues**

### **Potential Problems with the User Interface**

#### **1. Type checking**

When applications read or write records to the RODB, the type checking features of Ada are bypassed. Since the address list specifies only physical addresses, any datatype may be placed in these addresses. The RODB does check the types read with the attribute list, but there is no way of checking whether the address provided is indeed the address of a variable of the appropriate type. Thus, there is a need to type check the addresses in the address list with the types specified in the attribute list. Otherwise, there may be errors which could go undetected at compile-time and possibly even during the testing phase. For example, a trivial error which could easily occur is that the programmer accidentally interchanges two of the addresses in the address list. This would not be detected until possibly system integration time, when the wrong values would be retrieved and the system would malfunction.

#### **2. Nested records**

The current interface has a tendency to “flatten out” records, in the sense that it only provides for a two level hierarchy. Each entry in the attribute list consists of two parts, an object type and a field type. In fact, it is possible that an object may itself contain several records, each with several fields etc. It is even possible that the same field name may appear twice within the object, as part of different records. Ideally, there should be a way to describe fields which are part of more complex record structures. This deficiency may impose restrictions on the schemes used by the system designers for designing data types and creating field names.



### 3. Long repetitive lists

It is very inconvenient for the programmer to construct long address and attribute lists. The programmers will have a fair amount of work typing in long elements. The records consist of fields of arbitrary elements. The programmer has to be very careful to type in the corresponding number of entries. Typing in long lists is inconvenient and error-prone.

For example,

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  (APT_POSITION_RECORD.RIGHT_ASCENSION'address,  
   APT_POSITION_RECORD.DECLINATION.DEGREES'address,  
   APT_POSITION_RECORD.DECLINATION.MINUTES'address,  
   APT_POSITION_RECORD.DECLINATION.SECONDS'address);
```

Repetition of this form is very likely to occur, since very often programmers will want to read several fields from the same object. Some of these problems could be avoided if fields from the same record could be grouped together, in some form similar to the following:

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  (APT_POSITION_RECORD, (RIGHT_ASCENSION'address,  
                        DECLINATION.DEGREES'address,  
                        DECLINATION.MINUTES'address,  
                        DECLINATION.SECONDS.address));
```

The problem is particularly acute if the programmer wishes to read the entire object, as might often happen. In this case, the programmer must individually specify each field of the object, when it would be much simpler to just specify that the entire object needs to be read.

### 4. Modifiability

This interface is very sensitive to the object structure, the RODB storage scheme and the entire current RODB model. Since the space station is expected to have a long lifetime, it is likely that object definitions will change substantially, and even the actual RODB internal structure and interface design may change. Any such changes would require widespread changes to applications, affecting code in several modules in each application.

For example,

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  (APT_POSITION_RECORD.RIGHT_ASCENSION'address,  
   APT_POSITION_RECORD.DECLINATION.DEGREES'address,  
   APT_POSITION_RECORD.DECLINATION.MINUTES'address,  
   APT_POSITION_RECORD.DECLINATION.SECONDS'address);
```

where the common words are `APT_POSITION_RECORD` and `address`. Since each field is an address, the word `address` is redundant and could be eliminated. After factorization, a short list is produced:

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  APT_POSITION_RECORD. (RIGHT_ASCENSION,  
                        DECLINATION.DEGREES,  
                        DECLINATION.MINUTES,  
                        DECLINATION.SECONDS);
```

The repetitive nature of specifying information when writing several fields for the same record makes it inflexible and error-prone. If we need to access all the addresses in a record, it will be easier to avoid the repetitive specification. Using the above example and assume that there are only 4 fields in `APT_POSITION_RECORD`. Suppose we want access all the fields, then it is more convenient to perform this:

```
APT_POSITION_ADDRESS_ARRAY: constant STRDB.ADDRESS_ARRAY_T :=  
  APT_POSITION_RECORD.*
```

#### Write/Read

Using the same principle, it is convenient to write out or read an entire record. The programmer does not need to type out the long and repetitive lists to read/write out an entire record.

## Part III: Solutions

The preprocessor should be able to translate the programmer's input into the form used in the existing scheme. It should be able to perform datatype checking in order to eliminate any errors. Most of all, it should be convenient for the programmer to use.

## 1. Design 1

The programmer inputs the attribute-list and the corresponding address list. The preprocessor allows the use of shorter lists and eliminate the repetitive specification of the lists. The preprocessor will proceed to convert the programmer's input into output that will be used by the present scheme.

During the conversion, datatype checking is performed to check that there is no errors. This will prevent errors from occurring at run-time.

An example:

Suppose that the programmer's input is:

P\_attribute-list (R1.\*,R2.\*,R3.\*)

P\_address-list (I1,I2,I3)

Let's suppose that the format used in the present scheme:

attribute-list ((R1.F1), (R2.F21, R2.F22), (R3.F31, R3.F32, R3.F33))

address-list (&I1.F1, &I2.F21, &I2.F22, &I3.F31, &I3.F32, &I3.F33)

(where the symbol "&" is taken to mean "the address of")

The preprocessor will need to convert the attribute-list(programmer's) into the correct format. In order to perform this, we follow 2 steps:

1. Convert P\_attribute-list (R1.\*, R2.\*, R3.\*) into  
(R1(F1), R2(F21, F22), R3(F31,F32,F33))

2. Convert the result of 1. into  
attribute-list ((R1.F1), (R2.F21,R2.F22), (R3.F31,R3.F32,R3.F33))

The preprocessor will proceed to convert the P\_address-list into the correct format:

address-list (&I1.F1, &I2.F21, &I2.F22, &I3.F31, &I3.F32, &I3.F33)

During this conversion, the datatype would be checked. When there is no error, the output of the preprocessor would be the input for the present scheme with datatype checking done.

## Advantages

The programmer need not know about the more complex nature of the existing scheme due to the ease in communicating with the preprocessor. Hence, this scheme is more convenient and easier to use. Errors are less likely to happen and datatype checking is performed at compile-time. Hence, the safety of the system is ensured.

## Functionality

This scheme is application specific and has no impact on the amount of space used. The performance of this scheme will only be affected at compile-time, but it will be the same at run-time.

## 2. Design 2. (Not requiring a special syntax for the interface)

This design avoids modifying the existing interface. It only does type checking of the elements of the address list with the corresponding elements of the attribute list.

## Part IV: Comparison of Design 1 and Design 2.

Design 1 modifies the interface to avoid several of the problems identified. It keeps the programmer from having to deal with addresses etc. It solves the type checking problem by creating an interface where type errors cannot occur. However, it has the disadvantage that it requires the programmers to learn an interface different from the current one. It also requires a more complex preprocessor.

## Decision

In order to illustrate the power of a preprocessor, we decided to use design 2. It avoided the necessity for modifying the previously designed interface, and it kept the preprocessor construction simple.

The documented source code of the preprocessor is shown in **Appendix A**.

**Appendix A** provides the preprocessor design for the RODB Read/Write Interface. It was developed as part of the work on the "Complex data types" problem. In **Appendix A** the assumptions/limitations of the preprocessor are presented, then we include the source code, an example input and an example output.

# Performance Analysis

The RODB and STSV software subsystems consisted principally of Ada units running under the LYNX Operating System (DMS - OS) which is an UNIX-based Real-Time Operating System (RTOS). In brief, the mechanism of the data transfer between an application and DMS consists of two mechanisms of UNIX IPC. One is the IPC Message Queue and the other is IPC Shared Memory. Shared memory enlists another so-called IPC facility which are the UNIX IPC Semaphores as will be explained later. The IPC Message Queue facility of UNIX is shown diagrammatically in Figure 1 between the API and the Application Interface Program (AIP). The latter runs under the RODB Executive (REX). The other mechanism, the IPC Shared Memory (with semaphores), is where the objects are stored and can be accessed.

In the course of study by Dr.'s Leibfried, Zhao, Davari and Natarajan as well as by Research Assistant Libin Wu, it became apparent that the performance of the aforementioned facilities of IPC Message Queues and Shared Memory could present the most serious "bottlenecks" with the Embedded Data Processor (EDP).

## 1. IPC Message Queue Performance

IPC Message Queues are to be used by the DMS for the opening and closing of handles for read and write operations. They were originally also to be used to verify the validity of the "handle" information possessed by the DMS when an application wished to read or write an Queue facility was structured. Appendix B contains a brief description of the test program used to estimate the performance of IPC Message Queues. Procedure RODB\_Test6 is the main program which performed the test. To be even more conservative and realistic two separate programs should have been constructed but in this case RODB\_Test6 opened two message queues, spawned (forked) a child process which acted as a message recipient while the parent process acted as the message initiator. To simulate the RODB situation a second message acknowledging the initial message was sent by the child back to the parent through the other message queue. One supporting user-defined package for RODB\_Test6 is RODB\_Test\_Data containing interface pragmata for C system calls. The scenario of a "round trip" of a message was repeated 1000, 2000, 3000, 4000 and 5000 times. It was assumed that 5000 times would be the worst case for an EDP since there are to be two Standard Data Processor (SDP) Units on the SSF with one EDP in each SDP, (i.e. assuming an equal distribution of sensors/actuators for each unit). Originally it was common belief that most sensors attribute or set of attributes. With this in mind and considering the post-scrub estimate of something like 10,000 sensors and/or actuators a simple test of the performance of the Lynx IPC Message on the SSF would be "read" every second with some being read or written to several times per second. What the test showed, in the worst case, this objective could not be reached since it took over one second to send just 1000 round-trip messages and over six seconds to send 5000 round-trip mes-

sages. These times are even optimistic since the PS/2 used had a 20 MHz clock rate whereas the SSF space qualified EDP presently is only listed as only having a 16 MHz rate. Further, the test was run with no other jobs running on the PS/2 which will certainly not be the case in the operational system.

While this result is disconcerting, and was so reported, it is of less importance now since the software developer (IBM) presumably also realized the IPC Message Queue bottleneck and has redesigned the RODB component to contain a version number. Thus, the IPC Message Queues will be used only to open and close read and write handles and not to verify the handle validity for every read or write. The handle validity will be checked at "read time" by comparing the version number in the handle with that stored in the RODB component.

## 2. Shared Memory Access Performance

### 2(a). Test of Unimpeded Reads and Writes

Component writes respectively. The global data types are defined in package `RODB_Component_Data_Types`. That package consists only of data types and global constants plus interfaces to C user-defined functions and system functions and calls. The RODB Component is created as shared memory by initialization code in package `Rodb_Component` and three semaphores are created to "protect" the shared memory to enforce the Reader-Writer policy of mutual exclusion between readers and writers, as well as between two writers. Read access however is allowed by concurrent readers using a UNIX counting semaphore. The full effects of these semaphores are not used, however, in this test since the first program — `Rodbtst81` — only attempts to accomplish successive attribute reads from the RODB Component. Only the tests shown in Figure 2 called "Attribute - Read Flow" are performed and the two "write" semaphores are never locked. The data object being read consists of one of four of the primitive attributes, viz. Integer, Character, Boolean, or Float. The actual one chosen for each read is dependent upon the address of the portion of the component being read. This is much as it is in the proposed RODB Component. Before the read is initiated the reader semaphores are set or incremented. Specifically, the `Write_Lock` UNIX binary semaphore is tested, then if that is not locked the `Write_Intent` UNIX binary semaphore is tested and if that is not locked the `Read_Lock` UNIX counting semaphore is incremented. What may have biased the results is the fact that semaphore operations in the Lynx O.S. may no longer be atomic as is required in normal UNIX. To compensate for this, the priority was raised to 31 (the highest) just before the semaphores were tested and set and then the priority was lowered to the "normal" priority right after the actual testing and set operations of the semaphores. These semaphore operations are inside the shaded box in Figure 2. The actual code for these semaphore operations are shown in Appendix C-1 as C subprogram `readbeg` in file `readbeg.c`. After the actual RODB "read" the "readend" semaphore operations are performed. The `Read_Lock` UNIX counting semaphore is decreased. If the count goes to zero all "writers" sleeping on this event will be awakened. This is also shown in Figure 2. The code for this readend semaphore operation is shown in subprogram `readend` in file `readend.c`. Here too one can observe that raising the process priority to 31 before the semaphore operation and lowering it after the semaphore operation guarantees an

the file called rodbcomp1.dat. The actual Read\_Attrs or read-attributes program is shown in application defined package Rodb\_Component. The timing results are also shown in Appendix C-1 and for our "typical" figure of 5000 successive read events the total time was about 7.7 seconds. There are a few things which may bias these timing results. First, the two semaphore operations of testing Write\_Lock and Write\_Intent binary semaphores had to be done twice. The reason being an apparent deficiency in the Lynx "semop" algorithm in that even though the Write\_Lock and Write\_Intent semaphore testing is done as an array of semaphores the apparent operation does not follow the flow shown in Fig. 2. When the process is sleeping on a locked Write\_Intent semaphore and then awakened because of the Write\_Intent semaphore becoming unlocked the semop algorithm apparently does not retest the Write\_Lock semaphore as it should. To avoid this condition the two Write tests were repeated in order to get correct operation. This explains the array elements three (3) and four (4) in the semop array in function readbeg. A "proof" program illustrating this deficiency is shown later in this report and in Appendix E. Another minor item is the read itself, there is a small amount of additional computation required by the "if" statement which selects which type of element to read. Of course there is the requirement for raising the priority to guarantee atomiticity of the semop or semaphore operation system call. Thus both the "readbeg" and "readend" C functions use the fast\_setprio function to raise the priority of the operation while the semop system call is in progress and use it to lower it to normal afterwards.

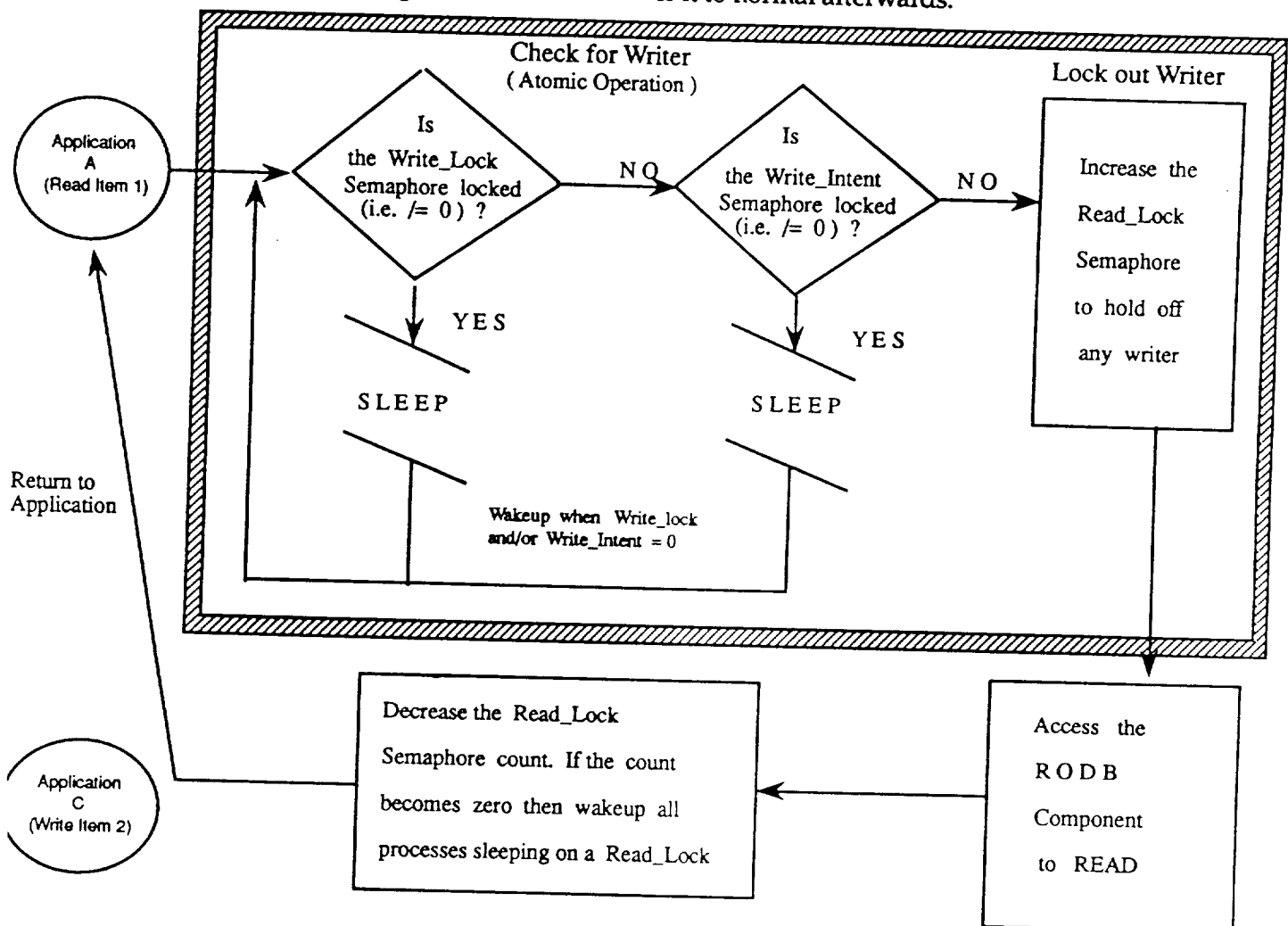


Figure 2. Attribute - Read Flow TFL/LW/CA Dec 91





is repeated for 1000, 2000, ... ,10000 times. The time, for the anticipated worst case, 5000 object writes took 8.78 seconds. The same comments about overhead apply to this experiment as for the "read" test.

**Appendix C-2** contains essentially the same programs as in **Appendix C-1** with the exception that the raising and lowering of process priority to guarantee the atomicity of operation of the semaphores is eliminated, i.e. commented out. The results of **Rodbtst91** for the same 5000 reads now drops to 2.57 seconds. These results are documented in **rodbcomp91.out** and **rodbcomp92.out** respectively in **Appendix C-2**. Corresponding performance time for the 5000 writes now takes 3.58 seconds. The drop in each case of over five (5) seconds indicates that even with the so-called **fast\_setprio** function call, the overhead for raising process priority is high. This would seem to indicate that if the Lynx semaphore system calls could support semaphore integrity without effectively disabling preemption of interrupts -- by raising priority -- it should be done. This test program is not truly valid since it reads four attributes rather than one but that is most probably more than compensated for by the fact that there would be more things occupying the processor than just **RODB** reads or writes in a practical situation. Clearly a one-second attribute read cycle for so many **RODB** reads with one 386 processor is hardly realistic.

In **Appendix C-3** the programs are the same as **C-1** but this time the **semaphore operations are commented out**. This is not a practical solution but it was run to try to determine the performance without the semaphore operations. The results for **RodbtstA1** the same "typical" 5000 reads was 5.0 seconds and for the writes the performance time was approximately 5.3 seconds. Raising priority is a way to guarantee mutual exclusion but at a high price. It effectively causes a "no-preemption" condition in the system guaranteeing atomicity of the reads and the writes just as might be expected with conventional UNIX where the Kernel cannot be preempted. The resulting performance penalty is too high for a real-time operating system.

In the **Appendix C** studies concurrency was not tested, so the performance time results of **Appendix C-1** almost exactly sum up to the results of **C-2** plus **C-3**. This was to be expected because of the absence of multiprocessing.

A more realistic program testing the mutual exclusion **raising the process priority** would do so **just before the read** and would **lower it right after the read** respectively rather than using this effective disabling of preemption twice to guarantee the atomicity of the semaphore operations. Similarly an analagous program to test writes would do the same thing. Such a program is shown in **Appendix C-4**. The results of this scenario with procedure **RodbtstB1** shows that the same benchmark of 5000 read cycles took 2.28 seconds and the corresponding 5000 write cycles took 2.21 seconds. This clearly shows that the system call to set a process priority, even the so-called "fast" changing-of-priority algorithm (i.e. **fast\_setprio**) is costly. Wherever possible dynamic priority changes should be avoided or at least minimized. Apparently, doubling of the number of **fast\_setprio** operations had the effect of raising the execution time over 100%. That is from 2.28 seconds for 5000 reads to 5.0 seconds as found in the programs in **Appendix C-3**.

Finally, in Appendix C-5 the programs RodbtstC1 (reads) and RodbtstC2 (writes) were modified versions of this series where raw "reads" and "writes" were accomplished without any critical section protection. That is to say no disabling of preemption nor semaphores were used. The resulting time for the benchmark 5000 read cycles or write cycles was on the order of 0.4 seconds. These operations were essentially non-competing in that the read cycles and write cycles were tested separately.

The tests in this section clearly indicate that the semaphore and preemption disabling and enabling operations are very costly in relationship to the basic read and write operations.

## 2(b). Test of Concurrent and Competing Reads and Writes

In Appendix D-1 procedure RodbtstD1 contains three tasks which all are of the same priority, two of them are reader tasks and one is a writer task. The support packages and programs are essentially the same as for Appendix B.

That is to say;

```
package Rodb_Component_Types,
package Rodb_Component,
function readbeg,
function readend,
function writebeg,
function writeend
function semsinit,
function semrmv,
function semprint,
and a new package called RODB_Test_Data1.
```

With the algorithm for each task in RodbtstD1 essentially duplicating the code used in Rodbtst81 or Rodbtst82 these Reader tasks competed with the Writer task for access to the RODB Component. In this test the readers accessed the RODB Component 2500 times each and the writer accessed the same Component 5000 times. In this case the writer "serviced" two readers. The elapsed time for all these reads and writes to complete was approximately 19 seconds. This time is the largest of the times of the three tasks competing to read or write. As one might suspect the writer logged the greatest elapsed time. It must be remembered that each task's priority was essentially raised to do the requisite semaphore operations and lowered after they were completed to guarantee the atomic operations of the semaphores and this "fast\_setprio" was done twice for each read or write.

In Appendix D-2 the same program as in Appendix D-1 was executed except that the code raising and lowering of the priorities to guarantee atomic semaphore operations was discarded (i.e. commented out). In that case the greatest elapsed time to access the RODB Component by two readers for 2500 reads each and for 5000 writes by the writer took 6.98 or approximately seven (7) seconds.

It is clear that the overhead of raising and lowering of priorities is costly since there is roughly a twelve (12) second difference in the performance times. The raising and lowering of the priorities almost tripled the maximum time for all of the tasks to finish. It seems clear that the price for compensating for the perceived deficiency in operating system services is too high. Effort could well be justified in correcting deficiencies rather than attempting "band-aid" compensation of them.

In Appendix D-3, for completeness, the code and corresponding data files are included for the case of no semaphores and no dynamic priority changes protecting the reads of the RODB Component. The "raw" reads and writes took on the order of two (2.13) seconds to complete 5000 each of concurrent reads and writes. When the number of reads was increased to 10000, the time to complete all the reads and writes took about four seconds (4.06). The main program in Appendix D-3 was renamed RodbtstF1 to distinguish its characteristics.

### 3. Test of Mutual Exclusion for the Reader-Writer Problem

Appendix E contains the code which is similar to that of the programs in Appendix C in that it simulates the Reader-Writer problem where multiple readers may read even the same attribute of an RODB object at the same time but readers and any writer must access the RODB mutually exclusively. That is to say a writer and a reader may not access the same attribute at the same (concurrent) time nor may two writers access the same attribute at the same time. In other words, by design, there is to be mutual exclusion between readers and writers and between different writers, but no mutual exclusion between different readers. The results of this test showed that there was indeed a violation of the mutual exclusion principle using the Lynx version of UNIX semaphores. These results cast a shadow on the results of the performance times of the competing reads and writes of Appendix D. These times might be too optimistic if readers and writer were allowed concurrent access to a given RODB component element when there should have been mutual exclusion. The main program is called Rodb\_Test7. This program does not use tasks since it only does one read or write. There are common packages with the Rodbtst80 and Rodbtst90 series used by Rodb\_Test7. These are the packages RODB\_Component\_Data\_Types and RODB\_Component. The C programs used are virtually identical to those used in Appendix D-2, (main program RodbtstE1). That is to say the array of semaphore operations used are the same but no set-priority system calls (i.e. setprio) are used.

What the programming system of Rodb\_Test7 does is to set up an RODB component shared memory segment and then permit the user to request either to read (reader) or to write (writer) to that shared memory. The semaphore operations are handled by the readbeg, writebeg, readend and writeend C function programs. The flow of these semaphore operations are shown in the Figure 2, the **Attribute Read Flow** and Figure 3, the **Attribute Write Flow**.

The shaded box regions shown in these diagrams indicate the operations which must be performed as an atomic transaction. What this translates to is the fact that if the kernel "sleeps" at any point

within these “boxes”, it must then retest all the semaphores from the beginning of the flow through the box in question when it resumes. This fact is also evident from Figure 4 which shows the algorithm for the **semop** system call, (see Bach [4]). The algorithm clearly shows the “go to start” or loop-back to the entry point of the algorithm upon any “wakeup”. **Rodb\_Test7**’s code permits it to act as either a reader or a writer process. The writer process code incorporates a pause statement in the **writebeg** function subprogram just after completing the semaphore operation and just before entering the “box” with the notation “Access the RODB Component to WRITE”. This causes the writer process to stall inside its critical section.

When that happens, any reader which is sleeping (waiting) for either the **Write\_Lock** or the **Write\_Intent** semaphore to be cleared should sleep for at least as long as the writer pauses, which is indefinite. What we found to happen is that any reader sleeping on the **Write\_Intent** semaphore will awaken when the **Write\_Intent** semaphore is cleared but **will not loop back to “start”** and test the **Write\_Lock** semaphore but will continue by increasing the **Read\_Lock** semaphore and enter its critical section to access the RODB Component. To show this, **Rodb\_Test7** must be invoked concurrently three times from three different terminals. The first process should be invoked as a reader, the second as a writer and the third as another reader. Under the scenario, the third process or second reader should block at the **Write\_Lock** semaphore and never finish because of the “pause” of the writer. Unfortunately what is observed is that the third process does finish even though the writer is still blocked at its pause statement. The full details of the test are described in the **read.me** file in **Appendix E**.

The results of this test lead to the assertion that if the reader-writer problem is to be properly addressed serious consideration must be given to correcting the deficiency of the **semop** algorithm in the Lynx Operating System. In fact just adjusting the algorithm to meet the normal AT&T UNIX System V performance may be insufficient for the real-time operations needed for Space Station Freedom. The **semop** algorithm relies upon the policy that the UNIX Kernel is non-preemptible this is counter productive to real-time operation [4]. Lynx has a preemptible Kernel [6]. If in Figure 2 , **Attribute Read Flow** the Kernel, running in behalf of a first reader process, is preempted, say just before the point where it will increase the reader counting semaphore to lock out a writer, when it resumed, it might not normally be cognizant that it had been preempted and the typical protocol would be to continue from whence it stopped execution. This could be disastrous if a preempting writer process also accessed the same RODB Component attribute. Mutual exclusion would again be violated. Clearly some reexamination of the structuring of semaphore algorithms in Lynx should be mandatory if the precept of a preemptible kernel is to be maintained and a set of semaphores which must maintain atomic operations is necessary.

```

algorithm semop /* semaphore operations*/
inputs: (1) semaphore descriptor; (2) array of semaphore operations
        (3) number of elements in the array;
output: start value of last semaphore operated on
{
    check legality of semaphore descriptor;
start:  read array of semaphore operations from user to kernel space;
        check permissions for all semaphore operations;
        for (each semaphore operation in the array)
        {
            if (semaphore operation is positive)
            {
                add "operation" to the semaphore value;
                if (UNDO flag is set on the semaphore operation)
                    update process undo structure;
                wakeup all processes sleeping (event semaphore value increases);
            }
            else if (semaphore operation is negative)
            {
                if ("operation" + semaphore value >= 0)
                {
                    add "operation" to semaphore value;
                    if (UNDO flag set)
                        update process undo structure;
                    if (semaphore value is 0)
                        wakeup all processes sleeping ( event
                                                semaphore value becomes 0);
                    continue;
                }
                reverse all semaphore operations already done on this system call
                    ( i.e. on previous iterations)
                if (flags specify not to sleep) return with error;
                sleep (event semaphore value increases);
                go to start; /* start loop from the beginning (nested transaction)*/
            }
        }
    else /* semaphore value is 0 */
    {
        if (semaphore value is not 0)
        { reverse all semaphore operations done this system call;
          if (flags specify not to sleep) return with error;
          sleep (event semaphore value == 0); go to start;
        }
    }
} /* for loop ends here */
update time stamps, process ID's ; /* semaphore operations succeeded */
return value of last semaphore operated on before call succeeded;
}

```

Figure 4. Algorithm for Semaphore Operation

## Summary of Observations and Suggestions

The study and analysis done under this so-called Complex Data Types project has of necessity been limited to attacking what seemed to be the salient features of the Data Management System for Space Station Freedom. Two distinct areas were investigated. One was the problem anticipated by the necessary circumvention of the strong typing features of Ada by the RODB. What was postulated was the concept of a preprocessor to capture the deficiencies of any code which might have interface irregularities. The structuring of a simple preprocessor showed what was anticipated. The MODB will need to have a strict naming convention in order to enable any automatic mechanism for correcting any program interface errors. The other area was performance of RODB STSV software in a real-time environment. With the limited resources at hand only the salient concepts of the anticipated major problem areas could be addressed. Two such areas were the IPC message queues for setting up RODB reads and writes and the mutual exclusion problem for protecting the integrity of critical data areas in the RODB Component itself. Performance problems were identified with both the UNIX IPC Message Queue and Semaphore constructs and a serious integrity problem was found with the semaphores.

The study results did identify potential problem areas for further investigation and indicate a direction for possible solutions to these problems. Even though final solutions were not defined, the identification of the deficiencies of resources which need to be used to satisfy the perceived

## References

1. McDonnell Douglas (MDC) Document No. H4350, "Software Preliminary Design Document (Data Management system Standard Services)," June 1990.
2. IBM No, 150A243 entitled "Software Detailed Design Document (Data Management System Standard Services)," November 1990.
3. McDonnell Douglas Document No. MDC H4193 APPENDIX I, entitled "Interface Requirement Document (Software) Appendix Interface Control Document (ICD) Data Management System (IBM)."
4. Bach, Maurice J. , The UNIX Operating System, Prentice-Hall, 1986.
5. Gallmeister, Wm. O. and Chris Lanier, "Early Experience with Posix 10034 and Posix 10034A," Proceedings Real-Time Systems Symposium, San Antonio, Texas, 4 - 6 , December 1991, IEEE-RTSS.
6. Singh, Inder M. and Mitch Bunnell, "LynxOS : UNIX Rewritten for Real-Time," Lynx Real-Time Systems Inc., Los Gatos, California.
7. Natarajan, S. , Gary Smith and T.F. Leibfried Jr., "Complex Data Types Project Preprocessor," Informal RICIS Document, 16 August 1991.
8. Leibfried, Jr., T.F., Sadegh Davari, Alfredo Pérez-Dávila and Libin Wu, "Complex Data Types Project DMS RODB Prototype Detailed Design Document (Preliminary)," RICIS Document Revision 6, 3 July 1991.
9. Leibfried, Jr., T. F. and Libin Wu, "DMS RODB Preliminary Design Document (Draft) of DMS Team (UHCL)," Informal RICIS Document Revision 9, 27 April 1991.
10. Leibfried, Jr., T.F. and Libin Wu, "DMS RODB Requirements (Draft) DMS Team (UHCL)," RICIS Informal Document Revision 8, 3 April 1991.





---

**ABBREVIATED APPENDICES**  
**APPENDICES A,B,C-1, D-1, E**



## **Appendix A**

### **Ada Preprocessor**



-----  
--  
-- Preprocessor Version 0.5  
--  
-----

--  
-- Changes from version 0.4  
--

-- User enters input and output filenames.  
--  
-- Ignores comments better (hopefully completely)  
--  
-- List\_pack doesn't allocate storage until actually needed.  
--  
-----

-- Assumptions/limitations:  
--

- 1. Input and Output must be different files.  
--
- 2. Correct Syntax. (input.ada must be a compilable Ada program)  
--
- 3. Flat name space in input.ada  
--
- 4. The read\_open calls and all of its associated attribute\_read calls  
-- are in input.ada.  
--
- 5. The specification and body of read open and attribute read are  
-- given in another routine (ie: 'with'ed from a package)  
--
- 6. Each read\_open will use a distinct handle.  
--
- 7. Each attribute\_read uses only one handle (not a list).  
--
- 8. The handle in an attribute\_read is the exact name given in  
-- corresponding read\_open.  
--
- 9. The syntax of the read\_open and attribute\_read is just like  
-- that in the CDR document. (except no list of handles).  
--
- 10. This routine produces new Ada code, which must then be seperately  
-- compiled.  
--
- 11. package 'dummy\_pack' is created. (decreased name space)  
--
- 12. A MODB routine is simulated in the preprocessor - when  
-- given the name of an attribute, returns the type of that  
-- attribute.  
--
- 13. The first begin in the file is the begin of the main program.  
-- This was necessary since the best place to put dummy\_pack just before  
-- it. (there are ways around this - as well as several of the above)  
--  
-----  
-----

-----  
--  
-- This is the simulated MODB package and function call  
--  
-----

-----  
package MODB\_package is  
function Get\_type (What : in String) return String;  
end MODB\_package;

```

package body MODB_package is
-----
-- when passed an attribute name, return the corresponding type
-----
function Get_type (What : in String) return String is
begin
    if (What = "AP TELESCOPE.X_POSITION") then
        return "X POSITION T";
    elsif (What = "AP TELESCOPE.Y_POSITION") then
        return "Y POSITION T";
    elsif (What = "AP TELESCOPE.SHUTTER") then
        return "SHUTTER POSITION T";
    elsif (What = "AP TELESCOPE.RIGHT_ASC") then
        return "RIGHT ASCENSION T";
    elsif (What = "AP TELESCOPE.DECLINATION") then
        return "DECLINATION T";
    elsif (What = "AP TELESCOPE.DEC DEGREES") then
        return "DECLINATION DEGREES T";
    elsif (What = "AP TELESCOPE.DEC MINUTES") then
        return "DECLINATION MINUTES T";
    elsif (What = "AP TELESCOPE.DEC SECONDS") then
        return "DECLINATION SECONDS T";
    elsif (What = "AP TELESCOPE.STATE") then
        return "STATE T";
    else
        return "Unknown_type";
    end if;
end Get_type;
end MODB_package;

-----
-- package LIST_PACK
--
-- This is a generic list package that must be instantiated with
-- A data type.
--
-- Unconstrained array type 'list_type' is made available as a
-- private type. The 'wither' should create a list of this
-- type and pass it back as a parameter to the calls in this
-- package. It was done this way so that several lists can be worked on
-- by each instantiation, while still keeping the workings of the
-- lists private.
-----

with Text_io; use Text_io;
Generic
    type Item_type is private;
package List_pack is

--
-- This is the type that is used by 'wither' to create a list.
--
type List_type(Max: Natural) is private;

function Size_of(Lister : in List_type) return Integer;
function Get_Item(Offset : Integer; Lister : in List_type) return Item_type;
procedure Add_item(Item: in Item_type; Lister : in out List_type);
procedure Update_item(Item: in Item_type; Lister : in out List_type;
    Offset : Natural);

private
    type Item_ptr_type is access Item_type;

```

```

type List_array is array(Integer range <>) OF Item_ptr_type;
type List_type(Max: Natural) is
  record
    List : List_array(0..Max);
    Num_in : Integer := 0;
  end record;

end List_pack;

-----
-- Body
-----
package BODY List_pack is

  -----
  -- Add to the list
  -----
  procedure Add_item(Item: in Item_type; Lister : in out List_type) is
  begin
    if Lister.Num_in = Lister.Max then
      Put(" ERROR: List_pack.Add_item: Lister of items is full - didn't add");
    else
      Lister.List(Lister.Num_in) := new Item_type'(item);
      Lister.Num_in := Lister.Num_in + 1;
    end if;
  end Add_item;

  -----
  -- Update an Item
  -----
  procedure Update_item(Item: in Item_type; Lister : in out List_type;
    Offset : Natural) is
  begin
    if Lister.Num_in < Offset then
      Put(" ERROR: List_pack.Update_item: offset out of current range");
    else
      Lister.List(Offset - 1).all := Item;
    end if;
  end Update_item;

  -----
  -- Return (without) deleting, the item at a particular offset
  -----
  function Get_item(Offset : Integer; Lister : in List_type) return Item_type is
  begin
    if Lister.Num_in = 0 then
      Put(" ERROR: List_pack.Get_item: Lister of items is empty- can't retrieve");
    elsif Offset > Lister.Num_in then
      Put(" ERROR: List_pack.Get_item: not that many items - can't retrieve");
    else
      return Lister.List(Offset - 1).all;
    end if;
  end Get_item;

  -----
  -- Returns how many items are currently in the list
  -----
  function Size_of(Lister: in List_type) return Integer is
  begin
    return Lister.Num_in;
  end;
end List_pack;

-----
-----

```

```

-- package inout_pack
--
-- This is a package to handle all reading from the input file and
-- writing to the output file
-- Upon instantiation, the input and output files are opened
-- Right now, they are static filenames
--
-----
with Text_io; use Text_io;
package Inout_pack is
    --
    -- Supply types and max lengths for the 'words' returned each
    -- call to get_word
    --
    Max_name_len : constant := 80;
    subtype Name_len_type is Integer range 0..Max_name_len;
    subtype Name_type is String(1..Max_name_len);

    --
    -- Declare a word type to be a name type and a current length
    -- Didn't make it private because of the plethora of times
    -- the individual fields are used elsewhere.
    --
    type Word_type is
        record
            Name : Name_type;
            Len : Name_len_type;
        end record;

    function Get_next word return Word_type;
    procedure Put_f (What : in String);
    procedure Put_line_f (What : in String := "");
    procedure Reset_input;
    function Open_files(inn_fname,out_fname : string) return boolean;
end Inout_pack;

-----
package BODY Inout_pack is

    Innf : File_type;
    Outf : File_type;
    Lookahead : Character := Ascii.nul;    -- next character in the input

    --
    -- resets the input file back to the beginning of the file
    --
    procedure Reset_input is
    begin
        Reset(Innf);
        Lookahead := Ascii.nul;
    end;

    --
    -- Outputs a string to the output file
    --
    procedure Put_f (What : in String) is
    begin
        Put(Outf,What);
    end Put_f;

    --
    -- Outputs a string and new_line to the output file
    --

```



```

procedure Put_line_f (What : in String := "") is
begin
  Put_line(Outf,What);
end Put_line_f;

```

```

-----
-- Scans the input file. Returns the next 'word'. Entire comments
-- are passed back as words. Blanks, line feeds, etc. are passed
-- back as one character long words.
-- Passes back an ascii.nul and length of 0 on end-of-file
-- It finds a single word by stopping at blanks, etc. This stop
-- character is stored in the internal (to body) variable
-- called lookahead. Lookahead is then used as the first character
-- on the next call
-----

```

```

function Get_next_word return Word_type is

```

```

  Word : Word_type;
  Char : Character;
  Cnt : Integer;

```

```

--
-- The following to_upper was taken out of the Verdex library
--

```

```

type convert_t is array(character) of character;

```

```

to_upper: constant convert_t := (
  ascii.nul, ascii.soh, ascii.stx, ascii.etx,
  ascii.eot, ascii.enq, ascii.ack, ascii.bel,
  ascii.bs, ascii.ht, ascii.lf, ascii.vt,
  ascii.ff, ascii.cr, ascii.so, ascii.si,
  ascii.dle, ascii.dcl, ascii.dc2, ascii.dc3,
  ascii.dc4, ascii.nak, ascii.syn, ascii.etb,
  ascii.can, ascii.em, ascii.sub, ascii.esc,
  ascii.fs, ascii.gs, ascii.rs, ascii.us,
  ' ', '!', '"', '#', '$', '%', '&', '\'',
  '(', ')', '*', '+', '-', '.', '/',
  '0', '1', '2', '3', '4', '5', '6', '7',
  '8', '9', ':', ';', '<', '=', '>', '?',
  '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
  'X', 'Y', 'Z', '[', '\', ']', '^',
  '_', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
  'X', 'Y', 'Z', '{', '|', '}', '~', ascii.del);

```

```

-----
-- Gets the next character
-----

```

```

function Get_char return Character is

```

```

  Temp_char : Character;
begin
  if End_of_file(Innf) then
    return Ascii.nul;
  elsif End_of_line(Innf) then
    Skip_line(Innf);
    return Ascii.lf;
  else
    Get(Innf,Temp_char);
    return to_upper(Temp_char);
  end if;
end Get_char;

```

```

begin
  if (End_of_file(Innf)) AND (Lookahead = Ascii.nul) then
    Word_Len := 0;

```

```

    Lookahead := Ascii.nul;
else
    if Lookahead = Ascii.nul then
        Word.Name(1) := Get_char;
    else
        Word.Name(1) := Lookahead;
    end if;
    Word.Len := 1;

    --
    -- Found the start of a word, get rest of word
    --
    if (Lookahead in 'A' .. 'Z') then
        loop
            Char := Get_char;
            exit when (Char not in 'A'..'Z') AND
                (Char not in '1'..'9') AND (Char /= '_');
            Word.Len := Word.Len + 1;
            Word.Name(Word.Len) := Char;
        end loop;
        Lookahead := Char;

        --
        -- Look to see if it is a comment, if so get entire comment
        -- To do this, have to read next character. if it is not
        -- a minus, pass the first back this time as word(1) and
        -- store the second in Lookahead
        --
        elsif (Lookahead = '-') then
            Char := Get_char;
            if (Char = '-') then
                Word.Len := 2;
                Word.Name(Word.Len) := Char;
                get_line(Innf, Word.Name(3)..Word.Name'last), Cnt);
                Word.Len := Cnt;
                Lookahead := Ascii.lf;
            else
                Lookahead := Char;
            end if;

            --
            -- else just pass that one character back as word(1) and get
            -- next character to the lookahead
            --
        else
            Lookahead := Get_char;
        end if;
    end if;

    return Word;
end Get_next_word;

function Open_files(inn_fname, out_fname : string) return boolean is
begin
    Open (Innf, in_file, inn_fname);
    Create (Outf, out_file, out_fname);
    return true;
exception
    when others => return false;
end;
end Inout_pack;

```

---



---

```

-- procedure PROCESS
--
-- This is the main preprocessor routine
--
-----
-----
with Text_io; use Text_io;
with List_pack;                -- Generic list package

with Inout_pack; use Inout_pack; -- Package to handle all I/O
                                -- includes definitions for Words, etc.
                                -- 'Used' it because it is used so often
with MODB_Package;

procedure Preprocess is

--
-- These set upper bounds on the number of read_opens, attribute_reads
-- and the number of components in an attribute or address array
--
Max_num_opens : constant Integer := 20;
Max_num_reads : constant Integer := 100;
Max_items : constant Integer := 20;
Dummy_pack_name : constant String := "dummy_pack";

--
-- Set up a list of Words so that it can be used to hold the parameters
-- to the read_opens and attribute reads
--
package Word_pack is NEW List_pack(Word_type);

--
-- Set up a list of read_open(s), each of which has a Handle name,
-- attribute list name, and a list to store the actual fields in
-- the attribute list - filled in on second pass
--
Type Open_type is record
  Handle : Word_type := ((others => ' '),0);
  Ary_id : Word_type := ((others => ' '),0);
  Ary : Word_pack.List_type(Max_items);
end record;
package Open_pack is NEW List_pack(Open_type);
Open_list : Open_pack.List_type(Max_num_opens);

--
-- Set up a list of attribute_read(s), each of which has an
-- address list name and a list to store the actual fields in
-- the address list - filled in on second pass
--
Type Read_type is record
  Ary_id : Word_type := ((others => ' '),0);
  Ary : Word_pack.List_type(Max_items);
end record;
package Read_pack is NEW List_pack(Read_type);
Read_list : Read_pack.List_type(Max_num_reads);

-----
-- Search through the list of attribute_reads for an address parameter
-- that is equal to Word. Return its Index if found, zero otherwise
-----
function Srch_read_Ary_id(Word : Word_type) return Integer is
  Temp_read : Read_type;
begin
  FOR Index in 1..Read_pack.Size_of(Read_list) loop

```

```

Temp_read := Read_pack.Get_item(Index, Read_list);
if (Temp_read.Ary_id.Len = Word.Len)
    and (Temp_read.Ary_id.Name(1..Word.Len)
        = Word.Name(1..Word.Len)) then
    return Index;
end if;
end loop;
return 0;
end;

```

```

-----
-- Search through the list of read opens for an attribute parameter
-- that is equal to Word. Return its Index if found, zero otherwise
-----

```

```

function Srch_open_Ary_id(Word:Word_type) return Integer is
    Temp_open : Open_type;
begin

```

```

    FOR Index in 1..Open_pack.Size_of(Open_list) loop
        Temp_open := Open_pack.Get_item(Index, Open_list);
        if (Temp_open.Ary_id.Len = Word.Len)
            and (Temp_open.Ary_id.Name(1..Word.Len)
                = Word.Name(1..Word.Len)) then
            return Index;
        end if;
    end loop;
    return 0;
end;

```

```

-----
-- Search through the list of read opens for a Handle parameter
-- that is equal to Word. Return its Index if found, zero otherwise
-----

```

```

function Srch_open_Handle(Word:Word_type) return Integer is
    Temp_open : Open_type;
begin

```

```

    FOR Index in 1..Open_pack.Size_of(Open_list) loop
        Temp_open := Open_pack.Get_item(Index, Open_list);
        if (Temp_open.Handle.Len = Word.Len)
            and (Temp_open.Handle.Name(1..Word.Len)
                = Word.Name(1..Word.Len)) then
            return Index;
        end if;
    end loop;
    return 0;
end;

```

```

-----
-- Outputs the complete dummy package with procedures for each valid
-- Handle along with the appropriate formal parameters
-----

```

```

procedure Write_package is
    Param_name : String(1..1);
    T_open : Open_type;
    T_read : Read_type;
    T_word : Word_type;
begin

```

```

    --
    -- output the package specification
    -- For each read_open, output a procedure spec for it using the
    -- Handle as the name of the procedure along with an extension
    -- include appropriate parameters for each field in its
    -- attribute list, starting with 'a' - name doesn't matter
    --

```

```

Put_line_f("-- THIS is A DUMMY package ADDED BY THE PREPROCESSOR --");
Put_line_f("package " & Dummy_pack_name & " is");

FOR i in 1..Open_pack.Size_of(Open_list) loop
  T_open := Open_pack.Get_item(i,Open_list);
  T_word := T_open.Handle;
  Put_line_f("    procedure " & T_word.Name(1..T_word.Len) &
                                                    "_dummy (");
  Param_name := "a";
  T_word := Word_pack.Get_item(1,T_open.Ary);
  Put_f("    " & Param_name & " : " &
        T_word.Name(1..T_word.Len));
  FOR k in 2..Word_pack.Size_of(T_open.Ary) loop
    Param_name(1) := character'succ(param_name(1));
    Put_line_f(";");
    T_word := Word_pack.Get_item(k,T_open.Ary);
    Put_f("    " & Param_name & " : " &
          T_word.Name(1..T_word.Len));
  end loop;
  Put_line_f(";");
end loop;
Put_line_f("end " & Dummy_pack_name & ";");

--
-- Output the Package body
-- For each read_open, output a Procedure body for it using the
-- Handle as the name of the procedure along with an extension
-- include appropriate parameters for each field in its
-- attribute list, starting with 'a' - name doesn't matter
-- only Statement in body should be null
--
Put_line_f("package body " & Dummy_pack_name & " is");
FOR i in 1..Open_pack.Size_of(Open_list) loop
  T_open := Open_pack.Get_item(i,Open_list);
  T_word := T_open.Handle;
  Put_line_f("    procedure " & T_word.Name(1..T_word.Len) &
                                                    "_dummy (");
  Param_name := "a";
  T_word := Word_pack.Get_item(1,T_open.Ary);
  Put_f("    " & Param_name & " : " &
        T_word.Name(1..T_word.Len));
  FOR k in 2..Word_pack.Size_of(T_open.Ary) loop
    Param_name(1) := character'succ(param_name(1));
    Put_line_f(";");
    T_word := Word_pack.Get_item(k,T_open.Ary);
    Put_f("    " & Param_name & " : " &
          T_word.Name(1..T_word.Len));
  end loop;
  Put_line_f(") is");
  Put_line_f("    begin");
  Put_line_f("        null;");
  Put_line_f("    end;");
end loop;
Put_line_f("end " & Dummy_pack_name & ";");
Put_line_f;

end;

-----
-- Outputs a call to the appropriate dummy procedure from the read_opens
-- uses Handle with extension as the procedure name and includes
-- actual paramters for each of the fields in the address list
-----
procedure Write_call(Handle : in Word_type;addr:in Word_type) is
  T_read : Read_type;
  T_word : Word_type;
  Index : Integer;

```

```

begin
  Put_line_f;
  Put_f("---- This is a call to the dummy procedure in dummy ");
  Put_line_f("package ----");
  Put_line_f(Dummy_pack_name & "." & Handle.Name(1..handle.Len) &
                                                    "_dummy (
Index := Srch_read Ary_id(addr);
T_read := Read_pack.Get_item(Index,Read_list);
T_word := Word_pack.Get_item(1,T_read.Ary);
Put_f("      " & T_word.Name(1..T_word.Len));
FOR i in 2..Word_pack.Size_of(T_read.Ary) loop
  Put_line_f(",");
  T_word := Word_pack.Get_item(i,T_read.Ary);
  Put_f("      " & T_word.Name(1..T_word.Len));
end loop;
Put_line_f(",");
end;

```

```

-----
-- Scan the input for attribute_read and read_open
-- Get the names of the attribute and address arrays
-----

```

```

procedure Pass_one (Open_list : in out Open_pack.List_type;
                    Read_list : in out Read_pack.List_type) is

```

```

  Word : Word_type := ((others => ' '),0);
  T_open : Open_type;
  T_read : Read_type;
  State: Integer range 0..4 := 0;

```

```

begin
  loop
    Word := Get_next_word;
    exit when Word.Len = 0;
    if (Word.Name(1) /= ' ') and
       (Word.Name(1) /= Ascii.ht) and
       (Word.Name(1) /= Ascii.lf) and
       (not ((Word.Len > 2) and then
            ((Word.Name(1) = '-') and (Word.Name(2) = '-')))) then
      case State is
        when 0 =>
          if (Word.Name(1..Word.Len) = "READ_OPEN") then
            State := 1;
            T_open.Ary_id := ((others => ' '),0);
            T_open.Handle := ((others => ' '),0);
          elsif (Word.Name(1..Word.Len) = "ATTRIBUTE_READ") then
            State := 3;
            T_read.Ary_id := ((others => ' '),0);
          end if;
        when 1 =>
          if (Word.Name(1) /= ',') then
            T_open.Ary_id.Name(1..Word.Len) := Word.Name(1..Word.Len);
            T_open.Ary_id.Len := Word.Len;
          else
            State := 2;
          end if;
        when 2 =>
          if (Word.Name(1) /= ',') then
            T_open.Handle.Name(1..Word.Len) := Word.Name(1..Word.Len);
            T_open.Handle.Len := Word.Len;
          else
            if Srch_open Handle(Word) = 0 then
              Open_pack.Add_item(T_open,Open_list);
            else
              Put_line("Error: same Handle used in two read_opens");
            end if;
          end if;
        -- State 3 is not reached in this version
      end case;
    end if;
  end loop;

```

```

        State := 0;
    end if;
    when 3 =>
        if (Word.Name(1) = ',') then
            State := 4;
        end if;
    when 4 =>
        if (Word.Name(1) /= ',') then
            T_read.Ary_id.Name(1..Word.Len) := Word.Name(1..Word.Len);
            T_read.Ary_id.Len := Word.Len;
        else
            if Srch_read_Ary_id(Word) = 0 then
                Read_pack.Add_item(T_read, Read_list);
            end if;
            State := 0;
        end if;
    end case;
end if;
end loop;
end Pass_one;

```

```

-----
-- Scan the input for the delcarations of attribute and address arrays
-- Store the fields in corresponding list
-----

```

```

procedure Pass_two (Open_list : in out Open_pack.List_type;
                    Read_list : in out Read_pack.List_type) is
    Word : Word_type := ((others => ' '), 0);
    T_word : Word_type := ((others => ' '), 0);
    Attr_type word : Word_type := ((others => ' '), 0);
    T_open : Open_type;
    T_read : Read_type;
    Index : Integer;
    State: Integer range 0..9 := 0;

begin
    Reset_input;
    State := 0;
    loop
        Word := Get_next_word;
        exit when Word.Len = 0;
        if (Word.Name(1) /= ' ') and
            (Word.Name(1) /= Ascii.ht) and
            (Word.Name(1) /= Ascii.lf) and
            (not ((Word.Len > 2) and then
                ((Word.Name(1) = '-') and (Word.Name(2) = '-')))) then
            case State is
                when 0 =>
                    if (Srch_open_Ary_id(Word) > 0) then
                        State := 1;
                        Index := Srch_open_Ary_id(Word);
                        T_open := Open_pack.get_item(Index, Open_list);
                    elsif (Srch_read_Ary_id(Word) > 0) then
                        State := 6;
                        Index := Srch_read_Ary_id(Word);
                        T_read := Read_pack.get_item(Index, Read_list);
                    end if;
                when 1 =>
                    if (Word.Name(1) = ':') then
                        State := 2;
                    elsif (Word.Name(1) = ',') then
                        State := 0;
                    end if;
                when 2 =>
                    if (Word.Name(1) = '(') then
                        State := 3;

```

```

        end if;
    when 3 =>
        if (Word.Name(1) = '(') then
            State := 4;
            T_word.Len := 0;
        end if;
    when 4 =>
        if (Word.Name(1) = ',') then
            T_word.Name(T_word.Len+1) := ',';
            T_word.Len := T_word.Len + 1;
        elsif (Word.Name(1) /= ')') then
            T_word.Name(T_word.Len+1..T_word.Len+Word.Len) :=
                Word.Name(1..Word.Len);
            T_word.Len := T_word.Len + Word.Len;
        else
            declare
                Attr_type : constant String :=
                    MODB_package.Get_type(T_word.name(1..T_word.Len));
            begin
                Attr_type_word.name(1..Attr_type'length) := Attr_type;
                Attr_type_word.Len := Attr_type'length;
            end;
            Word_pack.Add_item(Attr_type_word, T_open.Ary);
            Open_pack.Update_item(T_open, Open_list, Index);
            State := 5;
        end if;
    when 5 =>
        if (Word.Name(1) = ',') then
            State := 3;
        elsif (Word.Name(1) = ')') then
            State := 0;
        end if;
    when 6 =>
        if (Word.Name(1) = ':') then
            State := 7;
        elsif (Word.Name(1) = ',') then
            State := 0;
        end if;
    when 7 =>
        if (Word.Name(1) = '(') then
            State := 8;
            T_word.Len := 0;
        end if;
    when 8 =>
        if (Word.Name(1) /= '') then
            T_word.Name(T_word.Len+1..T_word.Len+Word.Len) :=
                Word.Name(1..Word.Len);
            T_word.Len := T_word.Len + Word.Len;
        else
            Word_pack.Add_item(T_word, T_read.Ary);
            Read_pack.Update_item(T_read, Read_list, Index);
            State := 9;
        end if;
    when 9 =>
        if (Word.Name(1) = ',') then
            T_word.Len := 0;
            State := 8;
        elsif (Word.Name(1) = ')') then
            State := 0;
        end if;
    end case;
end if;
end loop;
end Pass_two;

```



```

-----
-- Echo the input to the output along with the dummy package and
-- dummy procedure calls
-----
procedure Pass_three (Open_list : in out Open_pack.List_type;
                      Read_list : in out Read_pack.List_type) is
  Word : Word_type := ((others => ' '),0);
  T_Handle : Word_type := ((others => ' '),0);
  T_addr : Word_type := ((others => ' '),0);
  State: Integer range 0..5 := 0;

begin
  Reset_input;
  loop
    Word := Get_next_word;
    exit when Word.Len = 0;
    case State is
      when 0 =>
        if (Word.Name(1..Word.Len) = "BEGIN") then
          Write_package;
          State := 1;
        end if;
      when 1 =>
        if (Word.Name(1..Word.Len) = "ATTRIBUTE_READ") then
          State := 2;
        end if;
      when 2 =>
        if (Word.Name(1) /= ',') then
          T_Handle.Name(1..Word.Len) := Word.Name(1..Word.Len);
          T_Handle.Len := Word.Len;
        else
          State := 3;
        end if;
      when 3 =>
        if (Word.Name(1) /= ',') then
          T_addr.Name(1..Word.Len) := Word.Name(1..Word.Len);
          T_addr.Len := Word.Len;
        else
          State := 4;
        end if;
      when 4 =>
        if (Word.Name(1) = ';') then
          State := 5;
        end if;
      when 5 =>
        Write_call(T_handle,T_addr);
        State := 1;
    end case;
    if (word.name(1) = ascii.lf) then
      Put_line_f;
    else
      Put_f(Word.Name(1..Word.Len));
    end if;
  end loop;
  Put_line_f; -- forces a new_line at end of output file
end Pass_three;

-----
-- Main program
--
-- It works in three main passes:
-- 1. Scans the input and find each occurrence of 'open_read' and
--    'read_open'.
-- For each 'open_read', adds an 'open' item to the open_list
-- and stores the handle and name of the attribute array in that

```

```

--      item.
--      For each 'attribute_read', adds a 'read' item to the read_list
--      and stores the name of the address array in that item.
--
-- 2. Scans the input and find where each of the attribute arrays
--      and address arrays are declared.
--      For each attribute array found, update that item in the open list:
--      For each component in the declaration, add an entry in the
--      'ary' part of the open item that contains the type of that
--      component.
--      For each address array found, update that item in the read list:
--      For each component in the declaration, add an entry in the
--      'ary' part of the read item that contains the name of the
--      component without the address qualifier.
--
-- 3. Just before the begin of the main procedure, output a dummy
--      package that contains a procedure for each of the handles
--      in the open_list. The parameters to each array will have
--      dummy names but will be of the types obtained in the
--      attribute array.
--      For each of 'attribute_read': immediately after, place a call to
--      the appropriate dummy procedure. The parameters will be the
--      components found in the address array, except without the
--      address qualifier.
--
-----
begin
-----
--  Open input and Output files
-----
  declare
    inn_fname : string(1..Max_name_len);
    inn_size  : integer;
    out_fname : string(1..Max_name_len);
    out_size  : integer;
  begin
    loop
      put("Enter input file name -> ");
      get_line(inn_fname, inn_size);
      put("Enter output file name (not same as input) -> ");
      get_line(out_fname, out_size);
      exit when ((Inn_size /= out_size) or else
        (Inn_fname(1..inn_size) /= out_fname(1..out_size)));
      put_line("Input and Output must be different files");
    end loop;

    if Open_files(inn_fname(1..inn_size), out_fname(1..out_size))
    then
      Pass_one(Open_list, Read_list);
      Pass_two(Open_list, Read_list);
      Pass_three(Open_list, Read_list);
    else
      put_line("Error opening files");
    end if;
  end;

end Preprocess;

```

## **Appendix B**

### **IPC Message Queue Performance**



ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

			d		
			d		
			d		
r rrr	eeee	aaaa	ddd d	m m mm	eeee
rr r	e e	a	d dd	mm m m	e e
r	eeeeee	aaaaa	d d	m m m	eeeeee
r	e	a a	d d	m m m	e
r	e e	a aa	d dd	..	m m m e e
r	eeee	aaaa a	ddd d	..	m m m eeee

Job: read.me  
Date: Sat Mar 28 21:21:56 1992

This directory stores all the files to simulate the IPC MESSAGE QUEUE COMMUNICATION between processes. Two processes are created with one program using the fork system call. One is the sender (parent) and the other is the receiver (child). The sender sends a short message (request) to the receiver through one IPC Message Queue. The receiver receives the request and sends a response back to the sender through another IPC Message Queue. The sender receives the response message. This terminates one IPC communication. TESTING IS DONE FOR THE NUMBER OF SEND AND RECEIVING CYCLES FROM 1000 to 5000 in steps of 1000. The round-trip measured results are in file rodbcompl.out for the receiver and in file rodbcomp2.out for the sender.

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb		t		t
rr r	o o	d dd	bb b		t		t
r	o o	d d	b b		t	e e	t
r	o o	d d	b b		t	eeeeee	t
r	o o	d dd	bb b		t t	e e	t t
	oooo	ddd d	b bbb		tt	eeee	tt
						ssss	
						s s	
						ss	
						s s	
						ssss	

---

Job: rodb\_test\_data.ada  
Date: Sat Mar 28 21:30:25 1992

```

with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_Test_Data is

    -- Message queue system call interface
    function MSGGET(KEY : in integer;
                    FLAG : in integer) return integer;
    pragma INTERFACE(C, MSGGET);
    pragma INTERFACE_NAME(MSGGET, "msgget");
    function MSGSND(MSQID : in integer;
                    MSGP : in SYSTEM.address;
                    MSGSZ : in integer;
                    MSGFLG : in integer) return integer;
    pragma INTERFACE(C, MSGSND);
    pragma INTERFACE_NAME(MSGSND, "msgsnd");
    function MSGRCV(MSQID : in integer;
                    MSGP : in SYSTEM.address;
                    MSGSZ : in integer;
                    MSGTYP : in integer;
                    MSGFLG : in integer) return integer;
    pragma INTERFACE(C, MSGRCV);
    pragma INTERFACE_NAME(MSGRCV, "msgrcv");
    function MSGCTL(MSQID : in integer;
                    CMD : in integer;
                    BUFF : in SYSTEM.address) return integer;
    pragma INTERFACE(C, MSGCTL);
    pragma INTERFACE_NAME(MSGCTL, "msgctl");
end RODB_Test_Data;

```



ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb		t		t
rr r	o o	d dd	bb b		t		t
r	o o	d d	b b		ttttt	eeee	ssss
r	o o	d d	b b		t	e e	s s
r	o o	d dd	bb b		t	eeeeee	ss
r	oooo	ddd d	b bbb		t t	e e	s s
					tt	eeee	ssss
							ttttt
							t
							t
							t
							t t
							tt

```
-- This is the performance test for IPC message queue
with TEXT_IO, CALENDAR, SYSTEM, RODB_Test_Data, POSIX_UNSAFE_PROCESS_PR
use TEXT_IO, CALENDAR, SYSTEM, RODB_Test_Data, POSIX_UNSAFE_PROCESS_PR
with POSIX_PROCESS_IDENTIFICATION, POSIX_PROCESS_PRIMITIVES;
use POSIX_PROCESS_IDENTIFICATION, POSIX_PROCESS_PRIMITIVES;
procedure RODB_Test6 is
```

```
-- Constant definition
```

```
MSGKEY1    : constant integer := 99;
MSGKEY2    : constant integer := 100;
MTEXT_SIZE : constant integer := 500;
MSG_LEN    : constant integer := 10;
```

```
-- Data type definition
```

```
type MSGForm_Type is record
    MType : integer;
    MText : string(1..MTEXT_SIZE);
end record;
```

```
-- Package instantiation
```

```
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
```

```
-- Variable definition
```

```
My_PID      : POSIX_PROCESS_IDENTIFICATION.process_id;
My_Status   : POSIX_PROCESS_PRIMITIVES.termination_status;
Msgid1      : integer;
Msgid2      : integer;
Flag        : integer;
Number_Of_Times : integer;
Start_Time  : CALENDAR.time;
Finish_Time : CALENDAR.time;
Outfile     : file_type;
Sender_Msg  : MSGForm_Type;
Receiver_Msg : MSGForm_Type;
My_Msg      : string(1..MSG_LEN);
Your_Msg    : string(1..MSG_LEN);
My_Response : string(1..MSG_LEN);
Your_Response : string(1..MSG_LEN);
```

```
-- Exception definition
```

```
Msg_Exception : exception;
begin
```

```
-- Input the number of times from user
```

```
put("Enter the number of times: ");
INT_IO.get(Number_Of_Times);
```

```
-- Create two IPC Message Queues(request and response queues)
```

```
Msgid1 := RODB_Test_Data.MSGGET(MSGKEY1, 1023);
if Msgid1 = -1 then
    put_line("Error in MSGGET.");
    raise Msg_Exception;
end if;
Msgid2 := RODB_Test_Data.MSGGET(MSGKEY2, 1023);
if Msgid2 = -1 then
    put_line("Error in MSGGET.");
    raise Msg_Exception;
end if;
```

```

-- Create two processes (Sender and Receiver)
My_PID := POSIX_UNSAFE_PROCESS_PRIMITIVES.fork; -- fork a receiver child
if My_PID = POSIX_PROCESS_IDENTIFICATION.NULL_PROCESS_ID then -- Child
    Start_Time := CALENDAR.clock;
    for I in 1..Number_Of_Times loop
        Flag:=RODB_Test_Data.MSGRCV(Msgid1, Receiver_Msg'address, MSG_LEN, 1, 0);
        if Flag = -1 then
            put_line("Error in MSGRCV.");
            raise Msg_Exception;
        end if;
        My_Msg(1..MSG_LEN) := Receiver_Msg.MText(1..MSG_LEN);
        My_Response(1..MSG_LEN) := "Hello guys";
        Sender_Msg.MType := 1;
        Sender_Msg.MText(1..MSG_LEN) := My_Response(1..MSG_LEN);
        Flag := RODB_Test_Data.MSGSND(Msgid2, Sender_Msg'address, MSG_LEN, 0);
        if Flag = -1 then
            put_line("Error in MSGSND.");
            raise Msg_Exception;
        end if;
    end loop;
    Finish_Time := CALENDAR.clock;

    Delay 20.0; -- Wait for parent to manipulate the message queue

    -- Output the result to a file
    create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "Number_Of_Iterations Times");
    INT_IO.put(Outfile, Number_Of_Times);
    FIX_IO.put(Outfile, Finish_Time-Start_Time);
    new_line(Outfile);
    close(Outfile);

    POSIX_PROCESS_PRIMITIVES.exit_process; -- Child Exits
else -- Parent
    Start_Time := CALENDAR.clock;
    for I in 1..Number_Of_Times loop
        Your_Msg(1..MSG_LEN) := "Hi world!";
        Sender_Msg.MType := 1;
        Sender_Msg.MText(1..MSG_LEN) := Your_Msg(1..MSG_LEN);
        Flag := RODB_Test_Data.MSGSND(Msgid1, Sender_Msg'address, MSG_LEN, 0);
        if Flag = -1 then
            put_line("Error in MSGSND.");
            raise Msg_Exception;
        end if;
        Flag:=RODB_Test_Data.MSGRCV(Msgid2, Receiver_Msg'address, MSG_LEN, 1,0)
        if Flag = -1 then
            put_line("Error in MSGRCV.");
            raise Msg_Exception;
        end if;
        Your_Response(1..MSG_LEN) := Receiver_Msg.MText(1..MSG_LEN);
    end loop;
    Finish_Time := CALENDAR.clock;
end if;
POSIX_PROCESS_PRIMITIVES.wait_for_child(My_Status);

-- Output the result to a file
create(Outfile, out_file, "rodbcomp2.out",
    form=>"world=>read, owner=>read_write");

```

```

put_line(Outfile, "Number_Of_Iterations    Times");
INT_IO.put(Outfile, Number_Of_Times);
FIX_IO.put(Outfile, Finish_Time-Start_Time);
new_line(Outfile);
close(Outfile);

-- Remove the IPC Message Queues(request and response queues)
Flag := RODB_Test_Data.MSGCTL(Msgid1, 0, SYSTEM.NULL_ADDRESS);
if Flag = -1 then
    put_line("Error in MSGCTL.");
    raise Msg_Exception;
end if;
Flag := RODB_Test_Data.MSGCTL(Msgid2, 0, SYSTEM.NULL_ADDRESS);
if Flag = -1 then
    put_line("Error in MSGCTL.");
    raise Msg_Exception;
end if;

exception
    when Msg_Exception =>
        put_line("Program terminates abnormally.");
    when others =>
        put_line("Other exception in main program.");
end RODB_Test6;

```

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d	b					1
			d	b					11
			d	b					1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d dd	bb b	c c	o o	m m m	pp p		1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp		11111
							p		
							p		
							p		

Job: rodbcompl.out  
Date: Sat Mar 28 21:31:31 1992

Number_Of_Iterations	Times
1000	1.29565
Number_Of_Iterations	Times
2000	2.51501
Number_Of_Iterations	Times
3000	3.75342
Number_Of_Iterations	Times
4000	5.08582
Number_Of_Iterations	Times
5000	6.33374

11		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d	b					22222
			d	b					2
			d	b					
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		2
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p	222	
r	o o	d d	b b	c	o o	m m m	p p	2	
r	o o	d dd	bb b	c c	o o	m m m	pp p	2	
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp	222222	
							p		
							p		
							p		

Job: rodbcomp2.out  
Date: Sat Mar 28 21:31:44 1992

Number_Of_Iterations	Times
1000	1.29565
Number_Of_Iterations	Times
2000	2.52454
Number_Of_Iterations	Times
3000	3.75342
Number_Of_Iterations	Times
4000	5.08582
Number_Of_Iterations	Times
5000	6.33374



## **Appendix C-1**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**All Protection Mechanisms are Enabled as Semaphore Protection and Disabling of Preemption to Guarantee Atomic Semaphore Transitions in Place.**



			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

			d
			d
			d
r rrr	eeee	aaaa	ddd d
rr r	e e	a	d dd
r	eeeeee	aaaaa	d d
r	e	a a	d d
r	e e	a aa	d dd
r	eeee	aaaa a	ddd d

	m m mm	eeee
	mm m m	e e
	m m m	eeeeee
	m m m	e
..	m m m	e e
..	m m m	eeee

Job: read.me  
Date: Mon Mar 30 00:56:21 1992



This directory stores all the files to build up a RODB "attribute" components. The protection mechanism is that locking is set at the RODB level. During the lock setting, there is a prevention of preemption used to protect the semaphore test-and-lock operation to insure atomicity. This is done inside a C function by using the fast\_setprio system call. There is only one set of three UNIX semaphores in the whole system but a total of seven array operation on these three semaphores. Before actual READING, a set of five semaphore operations are imposed upon the three semaphores, two of which are repeated. The reason for the five (with two repeats) is to simulate what might have to be done if this were to be implemented with the current Lynx OS (where the kernel is preemptable) and no prevention of preemption were to be done; ( This is done in a later test). After actually reading the RODB one semaphore operation is imposed on the semaphores. Before actual WRITING there are two levels of operations: write desire and write lock. For write "desire" only one semaphore operation is imposed on the semaphores and for write "lock" an array of four semaphore operations are imposed on the semaphores. After actual writing, a set of two semaphore operations are imposed on the semaphores. A TEST IS DONE TO MEASURE HOW LONG IT TAKES FOR 1000 TO 100 READS AND WRITES. THE RESULTS ARE IN FILE rodbcomp1.dat for reads. THE RESULTS ARE IN FILE rodbcomp2.dat for writes. This test does not involve contention since the reads and writes are done in separate runs.

PRECEDING PAGE BLANK NOT FILMED



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d	b				88888	1
			d	b	t		t	8 8	11
			d	b	t		t	8 8	1 1
r rrr	oooo	ddd d	b bbb	ttttt	ssss	ttttt	8 8	1	
rr r	o o	d dd	bb b	t	s s	t	88888	1	
r	o o	d d	b b	t	ss	t	8 8	1	
r	o o	d d	b b	t	ss	t	8 8	1	
r	o o	d dd	bb b	t t	s s	t t	8 8	1	
r	oooo	ddd d	b bbb	tt	ssss	tt	88888	11111	

Job: rodbtst81.ada  
Date: Mon Mar 30 00:48:25 1992





```

-- This is the reading test program, (uncontested reads)
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure Rodbtst81 is

    -- Constant definitions
    ATTR_SIZE    : constant integer := 200;
    RESULT_SIZE  : constant integer := 10;

    -- Data type definition
    type Result_Type is record
        Loops : integer;
        Times : duration;
    end record;

    -- Package instantiation
    package INT_IO is new TEXT_IO.INTEGER_IO(integer);
    package FIX_IO is new TEXT_IO.FIXED_IO(duration);
    package RCDT renames RODB_Component_Data_Types;
    package RODBCP renames RODB_Component;

    -- Variable definitions
    Length          : integer;
    Number_Of_Times : integer;
    Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
    Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
    Start_Time      : CALENDAR.time;
    Finish_Time     : CALENDAR.time;
    Results         : array(1..RESULT_SIZE) of Result_Type;
    Outfile         : file_type;

begin
    RODBCP.Load_Comps("rodbcomp.dat"); -- load the test RODB
    Length := 1;
    Addr_List(1) := 0;
    Number_Of_Times := 1000; -- inner loop iterations initialization
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock; -- get the start time for inner loop
        for J in 1..Number_Of_Times loop
            RODBCP.Read_Attrs(Addr_List, Length, Attr_List); -- Read RODB
        end loop;
        Finish_Time := CALENDAR.clock; -- record the end time
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time); -- Store data
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;

    -- Output the result to a file now that test is over
    create(Outfile, out_file, "rodbcompl.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, " rodbcompl.dat ");
    put_line(Outfile, "Test NO  NO_Of_Iterations  Times");
    for I in 1..RESULT_SIZE loop
        INT_IO.put(Outfile, I, width => 5);
        INT_IO.put(Outfile, Results(I).Loops);
        FIX_IO.put(Outfile, Results(I).Times);
        new_line(Outfile);
    end loop;
    close(Outfile);
exception
    when others =>

```



```
put_line("Main program exception");  
end Rodbtst81;
```



```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb  cccc  oooo  m m mm  p ppp
rr  r  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p
r      o  o  d  d  b  b  c  o  o  m  m  m  p  p
r      o  o  d  d  b  b  c  o  o  m  m  m  p  p
r      o  o  d  dd  bb  b  c  c  o  o  m  m  m  pp  p
r      oooo  ddd d  b bbb  cccc  oooo  m  m  m  p ppp
                                     p
                                     p
                                     p

```

1  
11  
1 1  
1  
1  
1  
1  
11111

Job: rodbcompl.out  
Date: Mon Mar 30 00:57:09 1992



Test NO	NO_Of_Iterations	Times
1	1000	1.43866
2	2000	2.90466
3	3000	4.33374
4	4000	5.80927
5	5000	7.24799
6	6000	8.71387
7	7000	10.14307
8	8000	11.62805
9	9000	13.04773
10	10000	14.53265





ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

			d	b				88888	22222
			d	b	t		t	8	8
			d	b	t		t	8	8
r rrr	oooo	ddd d	b bbb	ttttt	ssss	ttttt	8	8	2
rr r	o o	d dd	bb b	t	s s	t	88888	222	
r	o o	d d	b b	t	ss	t	8	8	2
r	o o	d d	b b	t	ss	t	8	8	2
r	o o	d dd	bb b	t t	s s	t t	8	8	2
r	oooo	ddd d	b bbb	tt	ssss	tt	88888	2222222	

Job: rodbtst82.ada  
Date: Mon Mar 30 00:48:26 1992



```
-- This is the writing test program, uncontested writes
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure Rodbtst82 is
```

```
-- Constant definitions
```

```
ATTR_SIZE : constant integer := 200;
```

```
RESULT_SIZE : constant integer := 10;
```

```
-- Data type definition
```

```
type Result_Type is record
```

```
    Loops : integer;
```

```
    Times : duration;
```

```
end record;
```

```
-- Package instantiation
```

```
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
```

```
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
```

```
package RCDT renames RODB_Component_Data_Types;
```

```
package ROBCP renames RODB_Component;
```

```
-- Variable definition
```

```
Length : integer;
```

```
Number_Of_Times : integer;
```

```
Addr_List : RCDT.Pos_List_Type(1..ATTR_SIZE);
```

```
Attr_List : RCDT.Attr_List_Type(1..ATTR_SIZE);
```

```
Start_Time : CALENDAR.time;
```

```
Finish_Time : CALENDAR.time;
```

```
Results : array(1..RESULT_SIZE) of Result_Type;
```

```
Outfile : file_type;
```

```
begin
```

```
    ROBCP.Load_Comps("rodbcomp.dat"); -- load the RODB Components
```

```
    Length := 1;
```

```
    Addr_List(1) := 0;
```

```
    Attr_List(1) := (Type_ID => 0, Int_Value => 200);
```

```
    Number_Of_Times := 1000;
```

```
    for I in 1..RESULT_SIZE loop
```

```
        Start_Time := CALENDAR.clock; -- log the beginning
```

```
        for J in 1..Number_Of_Times loop
```

```
            ROBCP.Write_Attrs(Addr_List, Length, Attr_List); -- write
```

```
        end loop;
```

```
        Finish_Time := CALENDAR.clock; -- log the end
```

```
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
```

```
        Number_Of_Times := Number_Of_Times + 1000;
```

```
    end loop;
```

```
-- Output the result to a file now that test is over
```

```
create(Outfile, out_file, "rodbcomp2.out",
```

```
        form=>"world=>read, owner=>read_write");
```

```
put_line(Outfile, " rodbcomp2.out ");
```

```
put_line(Outfile, "Test NO NO_Of_Iterations Times");
```

```
for I in 1..RESULT_SIZE loop
```

```
    INT_IO.put(Outfile, I, width => 5);
```

```
    INT_IO.put(Outfile, Results(I).Loops);
```

```
    FIX_IO.put(Outfile, Results(I).Times);
```

```
    new_line(Outfile);
```

```
end loop;
```

```
close(Outfile);
```

```
exception
```



```
when others =>  
    put_line("Main program exception");  
end Rodbtst82;
```



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d dd	bb b	c c	o o	m m m	pp p
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

---

Job: rodb\_component\_.ada  
Date: Mon Mar 30 00:48:28 1992





```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List  : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List  : in      RCDT.Attr_List_Type);

    -- Print out the semaphore values
    procedure Print_Sems;

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end RODB_COMPONENT;

```



11		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d	b					22222
			d	b					2
			d	b					2
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		2
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p	222	
r	o o	d d	b b	c	o o	m m m	p p	2	
r	o o	d d	b b	c	o o	m m m	p p	2	
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp	2222222	
							p		
							p		
							p		

Job: rodbcomp2.out  
Date: Mon Mar 30 00:48:24 1992



rodbcomp2.out

Test NO	NO_Of_Iterations	Times
1	1000	1.73431
2	2000	3.57086
3	3000	5.30518
4	4000	7.09534
5	5000	8.88556
6	6000	10.66620
7	7000	12.40057
8	8000	14.22754
9	9000	15.92505
10	10000	17.76160



ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d dd	bb b	c c	o o	m m m	pp p
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

Job: rodb\_component\_data\_types\_.ada  
Date: Mon Mar 30 00:51:26 1992





```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type(Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY   : in integer;
                  SIZE  : in integer;
                  FLAG  : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID  : in integer;
                 SHMADDR : in system.address;
                 FLAG    : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```



```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

-- Semaphore system call and C function interface

```

function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

end RODB\_Component\_Data\_Types;



			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
r	oooo	ddd d	b bbb

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp p
c	o o	m m m	p p
c c	o o	m m m	pp p
cccc	oooo	m m m	p ppp
			p
			p
			p

Job: rodb\_component\_.ada  
Date: Mon Mar 30 00:54:23 1992



```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

  -- Package renaming
  package RCDT renames Rodb_Component_Data_Types;

  -- Exception definition
  Shm_Exception : exception;
  Shm_Outrange  : exception;
  Sem_Exception : exception;

  -- Read attributes from RODB components
  procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                       Length     : in      integer;
                       Attr_List  : in out RCDT.Attr_List_Type);

  -- Write attributes to RODB components
  procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length     : in      integer;
                        Attr_List  : in      RCDT.Attr_List_Type);

  -- Print out the semaphore values
  procedure Print_Sems;

  -- Load RODB components from a disk file
  procedure Load_Comps(Filename : in      string);

  -- Save RODB components to a disk file
  procedure Save_Comps(Filename : in      string);

  -- Shutdown the RODB components
  procedure Shutdown_Comps;

end RODB_COMPONENT;

```





ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d dd	bb b	c c	o o	m m m	pp p
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

---

Job: rodb\_component.ada  
Date: Mon Mar 30 00:48:30 1992

with TEXT\_IO, CALENDAR, SYSTEM, PREEMPTION\_CONTROL, Rodb\_Component\_Data\_Ty  
 use TEXT\_IO, CALENDAR, SYSTEM, PREEMPTION\_CONTROL, Rodb\_Component\_Data\_Ty  
 package body Rodb\_Component is

--Local variables

Shmid : integer;  
 Shmaddr : system.address;  
 Semid : integer;

-- Local subprograms

procedure Load\_Ints(Infile : in FILE\_TYPE);  
 procedure Load\_Chars(Infile : in FILE\_TYPE);  
 procedure Load\_Bools(Infile : in FILE\_TYPE);  
 procedure Load\_Flts(Infile : in FILE\_TYPE);  
 procedure Save\_Ints(Outfile : in FILE\_TYPE);  
 procedure Save\_Chars(Outfile : in FILE\_TYPE);  
 procedure Save\_Bools(Outfile : in FILE\_TYPE);  
 procedure Save\_Flts(Outfile : in FILE\_TYPE);

-- Read attributes from RODB components simulating array of handles read

procedure Read\_Attrs(Addr\_List : in RCDT.Pos\_List\_Type;  
                     Length : in integer;  
                     Attr\_List : in out RCDT.Attr\_List\_Type) is  
 Temp : system.address;  
 Flag : integer;  
begin  
 -- PREEMPTION\_CONTROL.DISABLE\_PREEMPTION; -- more efficient in C progr.  
 Flag := RCDT.READBEG(Semid);  
 -- PREEMPTION\_CONTROL.ENABLE\_PREEMPTION; -- in C program  
 if Flag = -1 then  
 raise Sem\_Exception;  
 end if;  
 for I in 1..Length loop  
 if (Addr\_List(I) < 0) or (Addr\_List(I) > RCDT.SHM\_SIZE-1) then  
 raise Shm\_Outrange;  
 end if;  
 Temp := Shmaddr + system.offset(Addr\_List(I));  
 if (Addr\_List(I) < RCDT.CHAR\_OFFSET) then  
 Attr\_List(I) := (Type\_ID => 0, Int\_Value => RCDT.FINT(Temp));  
 elsif (Addr\_List(I) < RCDT.BOOL\_OFFSET) then  
 Attr\_List(I) := (Type\_ID => 1, Char\_Value => RCDT.FCHAR(Temp));  
 elsif (Addr\_List(I) < RCDT.FLT\_OFFSET) then  
 Attr\_List(I) := (Type\_ID => 2, Bool\_Value => RCDT.FBOOL(Temp));  
 else  
 Attr\_List(I) := (Type\_ID => 3, Flt\_Value => RCDT.FFLT(Temp));  
 end if;  
 end loop;  
 -- PREEMPTION\_CONTROL.DISABLE\_PREEMPTION; -- again done in C program  
 Flag := RCDT.READEND(Semid); -- This is a C function  
 -- PREEMPTION\_CONTROL.ENABLE\_PREEMPTION;  
 if Flag = -1 then  
 raise Sem\_Exception;  
 end if;  
end Read\_Attrs;

-- Write attributes to RODB components again simulating array of handles

procedure Write\_Attrs(Addr\_List : in RCDT.Pos\_List\_Type;  
                     Length : in integer;  
                     Attr\_List : in RCDT.Attr\_List\_Type) is  
 Temp : system.address;

```

    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION; /* In C program for efficiency */
    Flag := RCDT.WRITEBEG(Semid); -- also a C function
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION; /* Also in C */
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1.. Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            RCDT.AINT(Temp, Attr_List(I).Int_Value);
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
        else
            RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
        end if;
    end loop;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION; /* In C program */
    Flag := RCDT.WRITEEND(Semid); -- A C function
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION; /* In C program */
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
    Flag : integer;
begin
    Flag := RCDT.SEMPRINT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--     Number_Of_Integers
--     Position1 Integer1
--     Position2 Integer2
--     ...
--     Number_Of_Characters
--     Position1 Character1
--     Position2 Character2
--     ...
--     Number_Of_Booleans
--     Position1 Boolean1
--     Position2 Boolean2
--     ...
--     Number_Of_Floats
--     Position1 Float1
--     Position2 Float2
--     ...
procedure Load_Comps(Filename : in string) is

```

```

Infile : FILE_TYPE;
Temp   : system.address;
Flag   : integer;
begin

    open(Infile, in_file, Filename);

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        RCDT.ACHAR(Temp, 'X');
    end loop;
    Load_Chars(Infile);

    -- Initialize RODB Boolean Component
    for I in 1..RCDT.BOOL_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        RCDT.ABOOL(Temp, true);
    end loop;
    Load_Bools(Infile);

    -- Initialize RODB Float Component
    for I in 1..RCDT.FLT_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        RCDT.AFLT(Temp, 0.0);
    end loop;
    Load_Flts(Infile);

    close(Infile);

    Flag := RCDT.SEMSINIT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;

exception
    when name_error =>
        put_line("File cannot be opened.");
        put_line("Loading components fails!");
    when data_error | end_error =>
        put_line("File format is incompatible.");
        put_line("Loading components fails!");
    when Sem_Exception =>
        put_line("Semaphore cannot be initialized.");
        raise Sem_Exception;
    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file
-- The structure of the disk file is as following:
--     Number_Of_Integers

```

```

--      Position1  Integer1
--      Position2  Integer2
--      ...
--      Number_Of_Characters
--      Position1  Character1
--      Position2  Character2
--      ...
--      Number_Of_Booleans
--      Position1  Boolean1
--      Position2  Boolean2
--      ...
--      Number_Of_Floats
--      Position1  Float1
--      Position2  Float2
--      ...
procedure Save_Comps(Filename : in      string) is
    Outfile : FILE_TYPE;
begin
    if Filename /= "" then
        create(Outfile, out_file, Filename,
               form=>"world=>read, owner=>read_write");

        Save_Ints(Outfile);
        Save_Chars(Outfile);
        Save_Bools(Outfile);
        Save_Flts(Outfile);
        close(Outfile);
    else
        Save_Ints(TEXT_IO.standard_output);
        Save_Chars(TEXT_IO.standard_output);
        Save_Bools(TEXT_IO.standard_output);
        Save_Flts(TEXT_IO.standard_output);
    end if;
exception
    when constraint_error =>
        put_line("RODB Components data collapsed.");
        put_line("Saving components fails!");
    when others =>
        put_line("Unknown exception.");
        put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
    Flag : integer;
begin
    Flag := RCDT.SHMDT(Shmaddr);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SEMSRMV(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Shutdown_Comps;

pragma page;

```

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
    Length      : Integer;
    Temp_Pos     : integer;
    Temp_Int     : integer;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        INT_IO.get(Infile, Temp_Int);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
        RCDT.AINT(Temp_Addr, Temp_Int);
    end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
    Length      : Integer;
    Temp_Pos     : integer;
    Temp_Char    : character;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        get(Infile, Temp_Char);           -- Skip a space
        get(Infile, Temp_Char);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
        RCDT.ACHAR(Temp_Addr, Temp_Char);
    end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
    Length      : Integer;
    Temp_Pos     : integer;
    Temp_Bool    : boolean;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        BOOL_IO.get(Infile, Temp_Bool);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
            raise Shm_Outrange;
        end if;
    end loop;
end Load_Bools;
```

```

    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

```

```

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is

```

```

    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

```

```

pragma page;

```

```

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in FILE_TYPE) is

```

```

    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

```

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in FILE_TYPE) is

```

```

    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Chars;

```

```

end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      eeee      aaaa      ddd d      b bbb      eeee      ggg g      cccc
rr  r      e  e      a      d  dd      bb  b      e  e      g  gg      c  c
r          eeeee      aaaaa      d  d      b  b      eeeee      g  g      c
r          e      a      a      d  d      b  b      e      g  g      c
r          e  e      a  aa      d  dd      bb  b      e  e      g  gg      ..  c
r          eeee      aaaa a      ddd d      b bbb      eeee      ggg g      ..  cccc
                                     g
                                     g
                                     gggg

```

Job: readbeg.c  
Date: Mon Mar 30 01:00:08 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protect
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START          = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform three semaphore operations, two of them twice (see read.me)
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK;      /* Wait for no more writ
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE;    /* Wait for no more writ
    one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK;      /* Wait for no more writ
    one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE;    /* Wait for no more writ
    one_of_n_readers[4] = RREAD_START;              /* Prevent writers in */
    flag = semop(semid, one_of_n_readers, 5);      /* Lock the critical sect
    if (flag == -1) {
        perror("readbeg fails: ");
    }

    /* Lower the priority to normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

                                d
                                d
                                d
r rrr      eeee      aaaa      ddd d      eeee      n nnn      ddd d      cccc
rr  r      e  e      a      d  dd      e  e      nn  n      d  dd      c  c
r          eeeee     aaaaa  d  d      eeeee  n  n      d  d      c
r          e  e      a  a      d  d      e  e      n  n      d  d      c
r          e  e      a  aa     d  dd     e  e      n  n      d  dd     ..  c
r          eeee      aaaa a     ddd d     eeee      n  n      ddd d     ..  cccc

```

Job: readend.c  
Date: Mon Mar 30 01:00:18 1992

```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);      /* Unlock critical section */
    if (flag == -1) {
        perror("readend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
      ttttt
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr  r      o  o      o  o
r          o  o      o  o
r          o  o      o  o
r          o  o      o  o
r          oooo      oooo

```

```

      i          t          b
      ii         t         b
      i          t         b
      i          t         b bbb
      i          t         e     e
      iii        tt        e     e     e     e     ggg g
      e     e     bb  b     e     e     g  gg
      eeeee     b    b     eeeee     g  g
      e     e     b    b     e     e     g  g
      e     e     bb  b     e     e     g  gg
      e     e     b bbb     e     e     ggg g
      e     e     e     e     ggg g
      ggg g
      gggg

```

Job: writebeg.c  
Date: Mon Mar 30 01:00:32 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/sched.h>
#include <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0};
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE     = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int flag;
    void perror();
    tid_t my_tid;
    int my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding reader */
    sole_writer[2] = WREAD_START;      /* preventing succeeding writer */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

                                     d
                                     d
                                     d
                                     d
w      w  r rrr  ii  ttttt  eeee  eeee  n nnn  ddd d
w w w rr  r  i  t  e  e  nn  n  d  dd
w w w r  i  t  eeeee  eeeee  n  n  d  d
w w w r  i  t  e  e  n  n  d  d
w w w r  i  t  t  e  e  n  n  d  dd  ..
  ww ww r  iii  tt  eeee  eeee  n  n  ddd d  ..

```

Job: writeend.c  
 Date: Mon Mar 30 01:00:58 1992

```

/* File: writeend.c This is write end subroutine to reset write protect
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;
    sole_writer[1] = WWRITE_UNLOCK;
    flag = semop(semid, sole_writer, 2);
    if (flag == -1) {
        perror("writeend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```



r rrr	oooo	oooo	t t tttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

ssss	eeee	m m mm	ssss	i		i	t t tttt	
s s	e e	mm m m	s s	ii	n nnn	ii	t	
ss	eeeeee	m m m	ss	i	nn n	i	t	
ss	e	m m m	ss	i	n n	i	t	
s s	e e	m m m	s s	i	n n	i	t t	..
ssss	eeee	m m m	ssss	iii	n n	iii	tt	..

Job: semsinit.c  
Date: Mon Mar 30 01:01:24 1992

```

/*File: semsinit.c This is semaphore init subroutine to initialize semap
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}

```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	tttt
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

ssss	eeee	m m mm	ssss	r rrr	m m mm	v v		cccc
s s	e e	mm m m	s s	rr r	mm m m	v v		c c
ss	eeeeee	m m m	ss	r	m m m	v v		c
ss	e e	m m m	ss	r	m m m	v v		c
s s	e e	m m m	s s	r	m m m	v v	..	c c
ssss	eeee	m m m	ssss	r	m m m	v	..	cccc

Job: semsrmv.c  
Date: Mon Mar 30 01:01:37 1992

```
/* File: semsrmv.c This is semaphore remove subroutine to remove semapho
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```

r rrr	oooo	oooo	t t tttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

ssss	eeee	m m mm	p ppp	r rrr	i	n nnn	t t tttt	
s s	e e	mm m m	pp p	rr r	ii	nn n	t	
ss	eeeeee	m m m	p p	r	i	n n	t	
ss	e e	m m m	p p	r	i	n n	t	
s s	e e	m m m	pp p	r	i	n n	t t	..
ssss	eeee	m m m	p ppp	r	iii	n n	tt	..
			p					
			p					
			p					

Job: semprint.c  
Date: Mon Mar 30 01:01:48 1992

```

/*File:semprint.c This is semaphore print subroutine to print semaphore
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semprint(semid)
    int semid;
{
    short outarray[3];
    int    flag;
    void    perror();
    int    i;

    flag = semctl(semid, 3, GETALL, outarray);
    if (flag == -1) {
        perror("semprint fails: ");
    }
    for (i=0; i<3; ++i) {
        printf("Semaphore %d has the value of %d\n", i, outarray[i]);
    }
    return(flag);
}

```

## **Appendix C-2**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**Semaphore Protection in Place but no Disabling of Preemption (i.e. no raising/lowering of priorities)**





```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

      d
      d
      d
: rrr  eeee  aaaa  ddd d
r  r  e   e  a   d  dd
.      eeeee  aaaaa  d  d
.      e      a   a  d  d
.      e   e  a   aa  d  dd
.      eeee  aaaa a  ddd d

      m m mm  eeee
mm m  m  e   e
m  m  m  eeeee
m  m  m  e
.. m  m  m  e   e
.. m  m  m  eeee

```

Job: read.me  
Date: Wed Apr 1 19:12:17 1992



## **Appendix C-3**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**Two Disabling and Re-enabling Pairs for each Read or Write. No Protection for Actual Reads or Writes.**



```

      t
      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      oooo  oooo  tt

```

```

      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr  r  e  e  a  d  dd
r      eeeee  aaaaa  d  d
r      e  a  a  d  d
r      e  e  a  aa  d  dd
r      eeee  aaaa a  ddd d

```

```

m m mm  eeee
mm m m  e  e
m m m  eeeee
m m m  e
.. m m m  e  e
.. m m m  eeee

```

Job: read.me  
Date: Sun Apr 5 20:20:45 1992

This directory stores all the files to build up RODB "attribute" component. The protection mechanism is absent for this test. There is no mechanism to assure mutual exclusion. Both the prevention-of-preemption by raising process priority and the semaphore operations were disabled. WHAT WE ARE TRYING TO MEASURE IS THE TIME REQUIRED FOR JUST THE "RAW" READS OR WRITES. THE TEST LOOPS TO MEASURE HOW LONG IT TAKES FOR 1000 to 10000 READS OR WRITES. THE RESULTS FOR READING ARE IN FILE rodbcomp1.out and the RESULTS FOR WRITING ARE IN FILE rodbcomp2.out. The reads and writes are non-competing.

## **Appendix C-4**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**Protected by Disabling Preemption before each Read or Write and Re-enabling After.**





## **Appendix C-5**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**No disabling of preemption  
No Semaphore Protection  
Just "raw" RODB Reads and Writes**



r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	tttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

r rrr	eeee	aaaa	ddd d	d	
rr r	e e	a	d dd	d	
r	eeeeee	aaaaa	d d	d	
r	e	a a	d d	d	
r	e e	a aa	d dd	d	
r	eeee	aaaa a	ddd d	d	

	m m mm	eeee
	mm m m	e e
	m m m	eeeeee
	m m m	e
..	m m m	e e
..	m m m	eeee

Job: read.me  
Date: Thu Apr 9 13:24:03 1992

This directory stores all the files to build up RODB "attribute" component. The protection mechanism is that locking is set at the RODB level. To achieve lock setting, a prevention-of-preemption system call is used. This is done inside a C function which has been called from Ada. The mechanism used is the fast setprio system call which is supposed to change priority expeditiously. WE ARE TRYING TO SEE HOW MUCH OVERHEAD THERE IS FOR THE PRIORITY CHANGE SYSTEM CALL. A TEST IS DONE TO MEASURE HOW LONG IT TAKES FOR 1000 TO 10000 READS AND WRITES (two priority raise/lower per Read or Write event). THE RESULT IS IN FILE rodbcomp1.out for reading. THE RESULT IS IN FILE rodbcomp2.out for writing.

## **Appendix D-1**

### **Concurrent (Competing) Reader and Writer Performance Test**

**Full Protection of Semaphores and RODB Component.**



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t t
r          oooo      oooo      tt

```

```

      d
      d
      d
r rrr      eeee      aaaa      ddd d
rr  r      e   e      a   d   dd
r          eeeee     aaaaa  d   d
r          e   e      a   a   d   d
r          e   e      a   aa  d   dd
r          eeee      aaaa a  ddd d

```

```

m m mm      eeee
mm m m      e   e
m m m      eeeee
m m m      e
.. m m m      e   e
.. m m m      eeee

```

Job: read.me  
Date: Sat Apr 11 23:09:58 1992





THIS IS A TEST OF CONCURRENT (COMPETING) READERS AND WRITERS.  
THERE ARE THREE TASKS RUNNING IN THIS SYSTEM: TWO READERS AND ONE WRITER.  
THEY ARE ACCESSING THE RODB COMPONENT WHICH IS PROTECTED BY A MECHANISM.  
IN RODBSTD1, ALL THE TASKS HAVE THE SAME PRIORITIES. THE RESULTS ARE IN  
FILES RODBCOMP1.OUT (RODBCOMP11.OUT AND RODBCOMP12.OUT). THESE RESULTS  
CORRESPOND TO THE INPUT FILES RODBCOMP1.IN (RODBCOMP11.IN AND RODBCOMP12.IN)

This directory stores all the files to build up RODB "attribute" components.  
The protection mechanism is that locking is set at the RODB level. During the  
lock setting, there is prevention of preemption. This is done inside the  
C function by using the fast\_setprio system call. There is only one set of  
three UNIX semaphores in the whole system. Before actual reading, a set of five  
semaphore operations are imposed on the semaphores. After a read completes one  
semaphore operation is imposed on a semaphore. Before actually writing,  
there are two levels of semaphore operations: write-desire and write-lock.  
For write-desire one semaphore operation is imposed (test&set) on a semaphore.  
Once the read-lock semaphore is unlocked, (i.e. last reader exits), then a  
writer can enter and lock out all new readers and any other following writer.  
Write-lock imposes a set of four semaphore operations on the semaphores.  
After actual writing, a set of two semaphore operations are imposed on the  
semaphores (i.e. unlock for readers or another writer). This system gives  
preferences to writers but readers actually reading lock out any writers  
that are waiting.



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

                        d  b
                        d  b
                        d  b
r rrr      oooo      ddd d  b bbb      ttttt      ssss      ttttt      DDDDD      1
rr  r      o  o      d  dd  bb  b      t      s      s      D  D      11
r          o  o      d  d  b  b      t      s      s      D  D      1 1
r          o  o      d  d  b  b      t      ss      t      D  D      1
r          o  o      d  dd  bb  b      t t      s      s      t t      D  D      1
r          oooo      ddd d  b bbb      tt      ssss      tt      DDDDD      11111

```

Job: rodbtstD1.ada  
 Date: Sat Apr 11 22:20:04 1992



```

-- This is the concurrent reading and writing test program with default priorit
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
with RODB_Test_Data1;
procedure RodbtstD1 is

```

```

-- Constant definitions

```

```

ATTR_SIZE      : constant integer := 200;
NUMBER_OF_TIMES1 : constant integer := RODB_Test_Data1.Number_Of_Times1;
NUMBER_OF_TIMES2 : constant integer := RODB_Test_Data1.Number_Of_Times2;
NUMBER_OF_TIMES3 : constant integer := RODB_Test_Data1.Number_Of_Times3;

```

```

-- Package instantiation

```

```

package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package ROBCP renames RODB_Component;

```

```

-- task declaration

```

```

task Reader1 is
    entry Finish;
end Reader1;
task Reader2 is
    entry Finish;
end Reader2;
task Writer is
    entry Finish;
end Writer;

```

```

-- Variable definition

```

```

Start_Time1 : CALENDAR.time;
Start_Time2 : CALENDAR.time;
Start_Time3 : CALENDAR.time;
Finish_Time1 : CALENDAR.time;
Finish_Time2 : CALENDAR.time;
Finish_Time3 : CALENDAR.time;
Result1      : duration;
Result2      : duration;
Result3      : duration;
Addr_List1   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
Addr_List2   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
Addr_List3   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
Attr_List1   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
Attr_List2   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
Attr_List3   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
Length1      : integer := 1;
Length2      : integer := 1;
Length3      : integer := 1;
Outfile      : file_type;

```

```

-- The body of task reader1

```

```

task body Reader1 is
begin
    Start_Time1 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES1 loop
        ROBCP.Read_Attrs(Addr_List1, Length1, Attr_List1);
    end loop;
    Finish_Time1 := CALENDAR.clock;
    Result1      := Finish_Time1 - Start_Time1;

```



```

    accept Finish;
exception
    when others =>
        put_line("Task Reader1 has an exception.");
end Reader1;

-- The body of task reader2
task body Reader2 is
begin
    Start_Time2 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES2 loop
        RODBCP.Read_Attrs(Addr_List2, Length2, Attr_List2);
    end loop;
    Finish_Time2 := CALENDAR.clock;
    Result2      := Finish_Time2 - Start_Time2;

    accept Finish;
exception
    when others =>
        put_line("Task Reader2 has an exception.");
end Reader2;

-- The body of task writer
task body Writer is
begin
    Start_Time3 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES3 loop
        RODBCP.Write_Attrs(Addr_List3, Length3, Attr_List3);
    end loop;
    Finish_Time3 := CALENDAR.clock;
    Result3      := Finish_Time3 - Start_Time3;

    accept Finish;
exception
    when others =>
        put_line("Task Writer has an exception.");
end Writer;

begin

    -- Terminate gracefully
    Reader1.Finish;
    Reader2.Finish;
    Writer.Finish;

    -- Write out the results
    create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "Task      Number_Of_Iterations      Times");
    put(Outfile, "Reader1  ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES1);
    FIX_IO.put(Outfile, Result1);
    new_line(Outfile);
    put(Outfile, "Reader2  ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES2);
    FIX_IO.put(Outfile, Result2);
    new_line(Outfile);
    put(Outfile, "Writer   ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES3);
    FIX_IO.put(Outfile, Result3);

```





```
new_line(Outfile);  
close(Outfile);  
  
end RodbtstD1;
```



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          o  o      o  o      tt
r          oooo      oooo

```

```

      d  b
      d  b
      d  b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr  r      o  o      d  dd      bb  b      c  c      o  o      mm m m      pp  p
r          o  o      d  d      b  b      c      o  o      m m m      p  p
r          o  o      d  d      b  b      c      o  o      m m m      p  p
r          o  o      d  dd      bb  b      c  c      o  o      m m m      pp  p
r          oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp
                                           p
                                           p
                                           p

```

Job: rodbcomp.dat  
Date: Sat Apr 11 22:27:31 1992



10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      oooo      ddd d      b bbb
rr  r      o      o      d      dd      bb      b
r          o      o      d      d      b      b
r          o      o      d      d      b      b
r          o      o      d      dd      bb      b
r          oooo      ddd d      b bbb

```

```

      cccc      oooo      m m mm      p ppp
c      c      o      o      mm m m      pp      p
c          o      o      m m m      p      p
c          o      o      m m m      p      p
c      c      o      o      m m m      pp      p
cccc      oooo      m m m      p ppp
p
p
p

```

---

Job: rodb\_component\_data\_types\_.ada  
Date: Sat Apr 11 22:28:55 1992

```

-- This package provides the constants, instantiated packages, system ca
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY   : in integer;
                  SIZE  : in integer;
                  FLAG   : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID  : in integer;
                 SHMADDR : in system.address;
                 FLAG    : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```



```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

```

-- Semaphore system call and C function interface

```

```

function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

```

end RODB_Component_Data_Types;

```

```

      t
      t
      ttttt
      t
      t
      t t
      tt
r rrr      oooo      oooo
rr  r      o   o      o   o
r          o   o      o   o
r          o   o      o   o
r          o   o      o   o
r          oooo      oooo

```

```

      d  b
      d  b
      d  b
      d  b bbb
      d dd bb  b
      d   d b   b
      d dd bb  b
      d dd
      ddd d b bbb
r rrr      oooo      ddd d
rr  r      o   o      d dd
r          o   o      d   d
r          o   o      d   d
r          o   o      d dd
r          oooo      ddd d

```

```

      cccc      oooo      m m mm
      c   c      o   o      mm m m
      c   c      o   o      m m m
      c   c      o   o      m m m
      c   c      o   o      m m m
      cccc      oooo      m m m
      p
      p
      p
      p
      p
      p

```

---

Job: rodb\_component\_.ada  
 Date: Sat Apr 11 22:29:11 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception  : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in      RCDT.Attr_List_Type);

    -- Print out the semaphore values
    procedure Print_Sems;

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end Rodb_Component;

```

```

r rrr      oooo      ddd d      b      cccc      oooo      m m mm      F
rr   r     o   o     d   dd     bb   b    c   c       o   o     mm m   m   F
r         o   o     d   d        b       b    c       o   o     m m m   m   F
r         o   o     d   d        b       b    c       o   o     m m m   m   F
r         o   o     d   dd     bb   b    c   c       o   o     m m m   m   F
r          oooo      ddd d      bbbb      cccc      oooo      m   m   m   F

```

---

Job: roldb\_component.ada  
Date: Sat Apr 11 22:29:31 1992

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READBEG(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READEND(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

Temp : system.address;
Flag : integer;
begin
  -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
  Flag := RCDT.WRITEBEG(Semid);
  -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
  if Flag = -1 then
    raise Sem_Exception;
  end if;
  for I in 1.. Length loop
    if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp := Shmaddr + system.offset(Addr_List(I));
    if (Addr_List(I) < RCDT.CHAR_OFFSET) then
      RCDT.AINT(Temp, Attr_List(I).Int_Value);
    elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
      RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
    elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
      RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
    else
      RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
    end if;
  end loop;
  -- delay 10.0;
  -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
  Flag := RCDT.WRITEEND(Semid);
  -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
  Flag : integer;
begin
  Flag := RCDT.SEMPRINT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--       Number_Of_Integers
--       Position1  Integer1
--       Position2  Integer2
--       ...
--       Number_Of_Characters
--       Position1  Character1
--       Position2  Character2
--       ...
--       Number_Of_Booleans
--       Position1  Boolean1
--       Position2  Boolean2
--       ...
--       Number_Of_Floats
--       Position1  Float1
--       Position2  Float2

```

```

--
...
procedure Load_Comps(Filename : in    string) is
    Infile : FILE_TYPE;
    Temp    : system.address;
    Flag    : integer;
begin
    open(Infile, in_file, Filename);

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        RCDT.ACHAR(Temp, 'X');
    end loop;
    Load_Chars(Infile);

    -- Initialize RODB Boolean Component
    for I in 1..RCDT.BOOL_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        RCDT.ABOOL(Temp, true);
    end loop;
    Load_Bools(Infile);

    -- Initialize RODB Float Component
    for I in 1..RCDT.FLT_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        RCDT.AFLT(Temp, 0.0);
    end loop;
    Load_Flts(Infile);

    close(Infile);

    Flag := RCDT.SEMSINIT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;

exception
    when name_error =>
        put_line("File cannot be opened.");
        put_line("Loading components fails!");
    when data_error | end_error =>
        put_line("File format is incompatible.");
        put_line("Loading components fails!");
    when Sem_Exception =>
        put_line("Semaphore cannot be initialized.");
        raise Sem_Exception;
    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
          form=>"world=>read, owner=>read_write");
    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```



pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Int    : integer;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
    RCDT.AINT(Temp_Addr, Temp_Int);
  end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Char   : character;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    get(Infile, Temp_Char);           -- Skip a space
    get(Infile, Temp_Char);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
    RCDT.ACHAR(Temp_Addr, Temp_Char);
  end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Bool   : boolean;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    BOOL_IO.get(Infile, Temp_Bool);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is
    Length : Integer;
    Temp_Pos : integer;
    Temp_Flt : float;
    Temp_Addr : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

pragma page;

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```

```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;

    -- Initialize the RODB Components
    Load_Comps("rodbcomp.dat");

end Rodb_Component;

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      eeee      aaaa      ddd d      b bbb      eeee      ggg g
rr  r      e      e      a      d      dd      bb      b      e      e      g      gg
r          eeeeee      aaaaa      d      d      b      b      eeeeee      g      g
r          e      e      a      a      d      d      b      b      e      e      g      g
r          e      e      a      aa      d      dd      bb      b      e      e      g      gg
r          eeee      aaaa a      ddd d      b bbb      eeee      ggg g      ..
                                           ggg g      ..
                                           g
                                           gggg

```

Job: readbeg.c  
Date: Sat Apr 11 23:40:14 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */ /* Guarantee atomic ops */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform three semaphore operations */ /* Two ops are repeated */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[4] = RREAD_START; /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 5); /* Lock the critical section */
    if (flag == -1) {
        perror("readbeg fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          oooo      oooo      tt

```

```

                        d
                        d
                        d
                        d
r rrr      eeee      aaaa      ddd d      eeee      n nnn      ddd d
rr  r      e      e      a      d      dd      e      e      nn      n      d      dd
r          eeeeee      aaaaa      d      d      eeeeee      n      n      d      d
r          e          a      a      d      d      e          n      n      d      d
r          e      e      a      aa      d      dd      e      e      n      n      d      dd
r          eeee      aaaa a      ddd d      eeee      n      n      ddd d      ..
                                           ..

```

```

c
c
c
c

```

Job: readend.c  
Date: Sat Apr 11 23:40:15 1992

```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);      /* Unlock critical section */
    if (flag == -1) {
        perror("readend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

          t
          t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r          oooo      oooo      tt

```

```

                                b
                                b
                                b
                                b bbb      eeee      ggg g
w      w  r rrr      ii      ttttt      e     e  bb   b  e     e  g   gg
w w w  rr   r      i      t      e     e  e     e  e     e  g   g
w w w  r          i      t      eeeee  b     b  eeeee  g   g
w w w  r          i      t      e     e  b     b  e     e  g   g
w w w  r          i      t t      e     e  bb   b  e     e  g   gg
  ww ww  r      iii      tt      eeee  b bbb      eeee  ggg g
                                g
                                g
                                gggg

```

Job: writebeg.c  
Date: Sat Apr 11 23:40:11 1992



```

/* File: writebeg.c This is the write begin subroutine to set protection */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/sched.h>
#include <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START = { 0, 1, 0}; /*lock out another writer*/
struct sembuf WWRITE_LOCK = { 1, 1, 0};
struct sembuf WWRITE_DESIRE = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0}; /* unlock write desire# */
/* #Guarantees writer progress */

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int flag;
    void perror();
    tid_t my_tid;
    int my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK; /* preventing succeeding readers */
    sole_writer[2] = WREAD_START; /* preventing succeeding writers */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request# */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }
    /* #Note cancelling the write-request allows another writer to lock it */

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

d  
 d  
 d  
 ld  
 d  
 d  
 ld  
 d

```
Job:  writeend.c
Date:  Sat Apr 11 23:40:13 1992
```

```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr      r  o      o  o      o      t
r          o      o  o      o      t
r          o      o  o      o      t
r          o      o  o      t      t
r          oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr      r  o      o  d      dd      bb      b      c      c      o      o      mm m m      pp      p
r          o      o  d      d      b          b      c          c      o      o      m m m      p      p
r          o      o  d      d      bb      b      c          c      o      o      m m m      pp      p
r          oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp
r                                     p
                                     p
                                     p

```

Job: rodbcomp.dat  
Date: Sat Apr 11 22:34:04 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10

0 A

1 B

2 C

3 D

4 E

5 F

6 G

7 H

8 I

9 J

10

0 false

1 false

2 false

3 false

4 false

5 false

6 false

7 false

8 false

9 false

10

0 100.0

1 200.0

2 300.0

3 400.0

4 500.0

5 600.0

6 700.0

7 800.0

8 900.0

9 1000.0

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t t
r          oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr  r      o   o      d dd      bb b      c   c      o   o      mm m m      pp p
r          o   o      d d      b   b      c   c      o   o      m m m      p p
r          o   o      d d      b   b      c   c      o   o      m m m      p p
r          o   o      d dd      bb b      c   c      o   o      m m m      pp p
r          oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp
                                           p
                                           p
                                           p

```

Job: rodbcomp11.in  
Date: Sat Apr 11 23:02:10 1992

2500 2500 5000

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
	oooo	oooo	tt

			d	b				
			d	b				
			d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m nm	p ppp	
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p	
r	o o	d d	b b	c c	o o	m m m	p p	
r	o o	d dd	bb b	c c	o o	m m m	p p	
r	oooo	ddd d	b bbb	cccc	oooo	m m m	pp p	
							p ppp	
							p	
							p	
							p	

Job: rodbcomp11.out  
Date: Sat Apr 11 23:02:22 1992



Task	Number_Of_Iterations	Times
Reader1	2500	12.98090
Reader2	2500	12.98090
Writer	5000	17.08582



5000 5000 5000

```

      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb  cccc  oooo  m m mm  p ppp
rr   r  o   o  d  dd  bb  b  c   c  o   o  mm m m  pp  p
r      o   o  d  d  b   b  c   c  o   o  m m m  p  p
r      o   o  d  d  b   b  c   c  o   o  m m m  p  p
r      oooo  ddd d  b bbb  cccc  oooo  m m m  p ppp
                                     p
                                     p
                                     p

```

Job: rodbcomp12.in  
Date: Sat Apr 11 23:02:41 1992

[illegible]

```
Job:  rodbcomp12.out
Date:  Sat Apr 11 23:03:17 1992
```

Task	Number_Of_Iterations	Times
Reader1	5000	25.99048
Reader2	5000	25.98096
Writer	5000	25.99048

## **Appendix D-2**

### **Concurrent (Competing) Reader and Writer Performance Test**

**No Prevention of Preemption but with Semaphore Protection of RODB Component.**





```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          oooo      oooo      tt

```

```

      d
      d
      d
r rrr      eeee      aaaa      ddd d
rr  r      e      e      a      d      dd
r          eeeeee      aaaaa      d      d
r          e          a      a      d      d
r          e      e      a      aa      d      dd
r          eeee      aaaa a      ddd d

```

```

m m mm      eeee
mm m m      e      e
m m m      eeeeee
m m m      e
m m m      e      e
m m m      eeee

```

```

..
..

```

Job: read.me  
Date: Mon Apr 20 17:46:59 1992

THERE ARE THREE TASKS RUNNING IN THE SYSTEM: TWO READERS AND ONE WRITER THEY ARE ACCESSING THE RODB COMPONENT WHICH IS PROTECTED BY A MECHANISM IN RODBTEST1, ALL THE TASKS HAVE THE SAME PRIORITIES. THE RESULTS ARE IN FILES RODBCOMP1.OUT (RODBCOMP11.OUT AND RODBCOMP12.OUT) for 5000 reads 5000 writes as well as 10000 reads and 5000 writes respectively. THESE CORRESPOND to INPUT FILES RODBCOMP1.IN (RODBCOMP11.IN and RODBCOMP12.IN

This directory stores all the files to build up RODB "attribute" component. The protection mechanism is that locking is set at the RODB level. During lock setting, there is NO prevention of preemption. Inside the C function the fast\_setprio system calls are commented out. There is only one set of UNIX semaphores in the whole system. Before actual reading, a set of five semaphore operations are imposed on the semaphores. After a read, one semaphore operation is imposed on a semaphore (decreasing reader count). Before writing there are two levels of semaphore operations which are imposed; namely, write-desire and write-lock. For write-desire only one semaphore operation is imposed on its semaphore (test&lock) and when that set and the last reader has finished, then, write-lock is set as one of a set of four semaphore operations imposed on the semaphores. After the writer finishes writing, a set of two semaphore operations are imposed on the semaphores (unlocking the RODB to allow readers or other writer in). THE INTENT IS TO DETERMINE THE EFFECT OF NOT USING PRIORITY RAISE/LOWER MANIPULATION WHICH FORMERLY GUARANTEED THE ATOMIC NATURE OF THE SEMAPHORE OPERATIONS. THIS TEST IS WITHOUT THAT COST.

## **Appendix D-3**

### **Simulated RODB**

#### **Concurrent (Competing), Reads and Writes**

**Prevention of Preemption Disabled**  
**Semaphore Protection Disabled**



```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr  r  e  e  a  d  dd
r      eeeee  aaaaa  d  d
r      e  a  a  d  d
r      e  e  a  aa  d  dd
r      eeee  aaaa a  ddd d

```

```

m m mm  eeee
mm m m  e  e
m m m  eeeee
m m m  e
m m m  e  e
m m m  eeee

```

Job: read.me  
Date: Wed Apr 15 19:28:49 1992

THERE ARE THREE TASKS RUNNING IN THE SYSTEM: TWO READERS AND ONE WRITER. THEY ARE ACCESSING THE RODB COMPONENT WHICH IS NOT NOW PROTECTED BY A MECHANISM. IN RodbtstF1 ALL THE TASKS HAVE THE SAME PRIORITY. THE RESULTS ARE IN FILES RODBCOMP1.OUT (RODBCOMP11.OUT AND RODBCOMP12.OUT). THE CORRESPONDING INPUT FILES ARE FILES RODBCOMP1.IN (RODBCOMP11.IN AND RODBCOMP12.IN).

This directory stores all the files to build up a RODB "attribute" component. The protection mechanism of locking at the RODB level is disabled. The prevention of preemption to protect the locking semaphores is also disabled. All of the protection was formerly done inside four C functions which used the fast\_setprio system call and the semop system call. There is one set of UNIX semaphores formerly used in the whole system. Now, all these system calls are disabled and no reader/writer protection is provided. While this would contribute to corrupt data items being read, the test was performed to see the time it would take for competing reads and writes in the "raw". Of course, now the overhead of calling the read and write beginning and ending functions is of little use, but they were left in the system to provide a method of isolating the costs of the protection mechanisms.

## **Appendix E**

### **Concurrent (Competing) Reads and Writes**

**Code for demonstrating the lack of mutual exclusion in a critical section for reading/writing to a simulated RODB component. The lack of mutual exclusion is presumably caused by the non-atomic nature of the semop system-call algorithm for an array of semaphores.**





```

      t
      t
      ttttt
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr  r      o  o      o  o
r          o  o      o  o
r          o  o      o  o
r          o  o      o  o
r          oooo      oooo

```

```

      d
      d
      d
      ddd d
      d  dd
      d  d
      d  d
      d  dd
      ddd d

r rrr      eeee      aaaa      ddd d
rr  r      e  e      a      d  dd
r          eeeee     aaaaa   d  d
r          e      a      a      d  d
r          e  e      a      aa   d  dd
r          eeee     aaaa a    ddd d

```

```

m m mm      eeee
mm m  m      e  e
m  m  m      eeeee
m  m  m      e
.. m  m  m      e  e
.. m  m  m      eeee

```

Job: read.me  
Date: Tue Apr 14 22:24:53 1992

This directory stores all the files to build up an RODB "attribute" compo. The protection mechanism is that locking is set at the RODB level. During lock setting, there is NO prevention of preemption. WE ASSUME A SYSTEM CALL IS AN ATOMIC ACTION. Assume there is one set of three UNIX semaphores in whole system. Before actual reading, a set of THREE semaphore operations be imposed on the semaphores. After actual reading one semaphore array operation will be imposed on the semaphores. Before actual writing, there are two levels of operations:

write-intent on one level and write-lock and read-lock on the other.

For write-intent only one semaphore operation will be imposed on the semaphore and for write-lock a set of four semaphore operations will be ordinarily imposed on the semaphores (i.e. including test of read-lock semaphore increase read-lock, set write-lock and clear write-intent semaphore).

BUT ON LYNX SEMAPHORE OPERATIONS, IF A PROCESS EXECUTING SEMOP IS PREEMPTED OR EVEN SLEEPS ON A SEMAPHORE EVENT, THERE IS NO GUARANTEE THAT SEMOP WILL RESUME AT THE VERY BEGINNING OF THE SEMOP ALGORITHM.

After actual writing, a set of two release-semaphore operations will ordinarily be imposed on the semaphores.

IN THIS PROGRAM, A PAUSE HAS BEEN INTRODUCED IN THE WRITER "CODE" SO THAT THESE TWO OPERATIONS (VIZ. WRITER LOCK AND READER LOCK) ARE NOT REINTERFERED. WHAT THIS TEST SHOWS IS FACT THAT IF THE READER HAS SLEPT ON THE WRITER INTENT SEMAPHORE THAT HAD BEEN ACTIVATED BY THE WRITER AND NOW HAD BEEN RELEASED, THE READER NOW IS AWAKENED, IT SHOULD, BUT WILL NOT GO BACK TO THE BEGINNING OF THE ALGORITHM TO CHECK THE STATUS OF THE WRITER LOCK SEMAPHORE. INSTEAD IT RESUMES WHERE IT LEFT OFF AND ONLY CHECKS THE WRITER INTENT SEMAPHORE WHICH HAS JUST BEEN CLEARED. HENCE, WE NOW HAVE BOTH A READER AND A WRITER IN THE CRITICAL SECTIONS AT THE SAME TIME.

To run this program --once compiled-- three terminals are required. Each terminal should log into the same account and execute rodb test7. When this is done a menu will appear for each process on its respective terminal screen. One process should choose option "4" to load the shared memory and press the enter key (i.e. <CR>). The question is then "asked" for the name of the file from which to load the contents of the shared memory. The simplest is to press "Enter" since there is a default data file which will be loaded by pressing the Enter key (i.e., <CR>). Once this is done the memory may be viewed by selecting option 3. Then, when the menu appears again, this first terminal should choose the option "1" to read a list of attributes followed by <CR>. When the question is asked "how many" the simplest answer is to type 1 <CR>. The question will then be raised as to the address to be "read", the simplest answer is to type the number "0" (without the quotes) BUT DO NOT PRESS "Enter". Then set up the next terminal by running another copy of rodb test7 and choose the option "2" to write a list of attributes <CR>. Then, when the question "asked" for "how many", the answer is to type 1 <CR>. As before the address "question" will be asked. Type in the number "0" as before followed by <CR>. The question will then be asked for the value to be written. You may type some integer such as 200 (the number 100 is the default at address 0). Again DO NOT TYPE <CR> or Enter !! Then start the third process from the THIRD terminal. This time choose the option "1" again to read. Again select option (1) attribute and again select address "0" BUT DO NOT TYPE <CR>. NOW, go back to terminal one (1) and press Enter (<CR>). Right after that (less than five seconds) press Enter (<CR>) on terminal two (2). After that (less than five but more than two seconds) press "Enter" on terminal three (3). Clearly the first process will encounter the

semaphores before process two and will lock out process two since it will increase the reader semaphore. Process two (2) will set the write-intent semaphore but will block on the reader semaphore. Process three (3) will test the write-lock semaphore and find it NOT set (Process one (1) is structured to stay in the "reader" critical section about ten seconds so the writer will be "locked out"). HOWEVER, Process three (3) will find the write-intent semaphore set so it will "sleep" on the event that the write-intent semaphore is "cleared". Meanwhile eventually -- after ten seconds -- the first reader finishes and "clears" the reader semaphore. Immediately the the second process (2) will wake to find the reader semaphore to be zero. This writer process (2) will then set the "write-lock" semaphore but will release the write-intent semaphore so as to allow higher priority readers to wake up and sleep on the "write-lock" semaphore rather than the "write-intent" semaphore (2) then enters the critical section to access the RODB Component. THIS IS WHERE THE PROBLEM APPEARS since process three (3) apparently does wake up BUT DOES NOT "GO TO START" and recheck the "write\_lock" semaphore and the "write-intent" semaphore but rather just continues on by (perhaps) not checking the "write-lock" semaphore but only the "write-intent" semaphore and finding it "clear" AND then it increases the "read-lock" semaphore and reads the shared memory. It is thus reading the shared memory at essentially shared memory. You will see the process three write out what it has read and terminate. Process two (2), which started before process three (3) is blocked at its "pause" statement just before accessing the shared memory. Process three (3) finishes even though it should be blocked from entering by the "write\_lock" semaphore set by process two (2).

Hence the Lynx semop algorithm does not meet the System V requirements and further even if it did in the non-preemptive case, it might fail under the real-time preemptive case. That is to say, if the process does not sleep but is preempted during the execution of algorithm semop when it resumed it would not know that it had been preempted and would continue on. Thus the same scenario could happen as above even if the reader process did not sleep but was preempted by an high-priority writer just before it increased the "read-lock" semaphore.

```

      t
      t
      ttttt
      t
      t
      t  t
      tt

r rrr      oooo      oooo
rr  r      o      o      o      o
r          o      o      o      o
r          o      o      o      o
r          o      o      o      o
r          oooo      oooo

```

```

      d      b
      d      b
      d      b
      ddd d  b bbb
      d  dd  bb  b
      d  d  b      b
      d  d  b      b
      d  dd  bb  b
      ddd d  b bbb

r rrr      oooo      ddd d
rr  r      o      o      d  dd
r          o      o      d  d
r          o      o      d  d
r          o      o      d  dd
r          oooo      ddd d

```

```

      t
      t
      ttttt      eeee      ssss      t
      t          e      e      s      s
      t          eeeee      ss
      t          e          ss
      t  t      e      e      s      s
      tt          eeee      ssss

```

---

Job: rodb\_test7.ada  
 Date: Tue Apr 14 22:25:16 1992

```

-----
--| This is main program for Test of Reader-Writer mutex problem for the RODB
-- Component. The RODB Component is represented by a small Shared Memory
-- Segment which is set up by functions and procedures in package RODBCP which
-- is a "rename" for package RODB_COMPONENT_DATA_TYPES. Most of these RODBCP
-- functions are Ada names for C library functions and UNIX System Calls.RODBC
-- operations and the accessing of the RODB Shared Memory segment.
use TEXT_IO, SYSTEM, RODB_Component_Data_Types, RODB_Component;

```

```

procedure Rodb_Test7 is

```

```

    ATTR_SIZE : constant integer := 20;
    package RCDT renames RODB_Component_Data_Types;
    package RODBCP renames RODB_Component;
    Length : integer;
    Filename : string(1..13);
    File_Len : integer;
    Addr_List : RCDT.Pos_List_Type(1..ATTR_SIZE);
    Attr_List : RCDT.Attr_List_Type(1..ATTR_SIZE);
    Choice : integer;

```

```

-- Input a list of addresses at the unit of bytes

```

```

procedure Input_Addr_List(Addr_List : in out RCDT.Pos_List_Type;
                          Length      : in      integer) is

```

```

begin

```

```

    for I in 1..Length loop
        put("Address number ");
        INT_IO.put(I, width=>3);
        put(":");
        INT_IO.get(Addr_List(I));
    end loop;
end Input_Addr_List;

```

```

-- Input a list of attributes according to their addresses

```

```

procedure Input_Attr_List(Attr_List : in out RCDT.Attr_List_Type;
                          Length      : in      integer;
                          Addr_List   : in      RCDT.Pos_List_Type) is

```

```

    An_Int : integer;
    A_Char : character;
    A_Bool : boolean;
    A_Flt  : float;
begin
    for I in 1..Length loop
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            put("Enter an attribute integer: ");
            INT_IO.get(An_Int);
            Attr_List(I) := (Type_ID => 0, Int_Value => An_Int);
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            put("Enter an attribute character: ");
            get(A_Char);
            Attr_List(I) := (Type_ID => 1, Char_Value => A_Char);
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            put("Enter an attribute boolean: ");
            BOOL_IO.get(A_Bool);
            Attr_List(I) := (Type_ID => 2, Bool_Value => A_Bool);
        else
            put("Enter an attribute float: ");
            FLT_IO.get(A_Flt);
            Attr_List(I) := (Type_ID => 3, Flt_Value => A_Flt);
        end if;
    end loop;
end Input_Attr_List;

```

```
-- Output a list of Attributes according to their addresses
procedure Output_Attr_List(Attr_List : in      RCDT.Attr_List_Type;
                           Length      : in      integer) is
```

```
begin
  for I in 1.. Length loop
    put("Attribute number");
    INT_IO.put(I, width => 3);
    put(" is ");
    case Attr_List(I).Type_ID is
      when 0 =>
        put("Integer: ");
        INT_IO.put(Attr_List(I).Int_Value);
      when 1 =>
        put("Character: ");
        put(Attr_List(I).Char_Value);
      when 2 =>
        put("Boolean: ");
        BOOL_IO.put(Attr_List(I).Bool_Value);
      when 3 =>
        put("Float: ");
        FLT_IO.put(Attr_List(I).Flt_Value);
      when others =>
        null;
    end case;
    new_line;
  end loop;
end Output_Attr_List;
```

```
begin
  loop
    put_line("1----Read a list of attributes");
    put_line("2----Write a list of attributes");
    put_line("3----print out the shared memory");
    put_line("4----Load the shared memory");
    put_line("0----Exit");
    put("Input your selection: ");
    INT_IO.get(Choice);
    skip_line;
    case Choice is
      when 0 =>
        exit;
      when 1 =>
        put("How many attributes do you want: ");
        INT_IO.get(Length);
        Input_Addr_List(Addr_List, Length);
        RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
        Output_Attr_List(Attr_List, Length);
      when 2 =>
        put("How many attributes do you want: ");
        INT_IO.get(Length);
        Input_Addr_List(Addr_List, Length);
        Input_Attr_List(Attr_List, Length, Addr_List);
        RODBCP.Write_Attrs(Addr_List, Length, Attr_List);
      when 3 =>
        put("Enter the filename to send to(none to screen): ");
        get_line(Filename, File_Len);
        RODBCP.Save_Comps(Filename(1..File_Len));
      when 4 =>
        put("Enter the filename to load from(none from rodbcomp.dat): ");
```

```

        get_line(Filename, File_Len);
        if (File_Len /= 0) then
            RODBCP.Load_Comps(Filename(1..File_Len));
        else
            RODBCP.Load_Comps("rodbcomp.dat");
        end if;
        when others =>
            put_line("Input error!");
        end case;
    end loop;
    RODBCP.Shutdown_Comps;
exception
    when Shm_Exception =>
        put_line("Shared memory not accessible.");
    when Shm_Outrange =>
        put_line("Shared memory out of range.");
    when Sem_Exception =>
        put_line("Semaphores not accessible.");
end Rodb_Test7;

```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
r	oooo	ddd d	b bbb

cccc	oooo	m m mm
c c	o o	mm m m
c	o o	m m m
c	o o	m m m
c c	o o	m m m
cccc	oooo	m m m

---

Job: rodb\_component\_data\_types\_  
Date: Tue Apr 14 22:25:41 1992



```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY   : in integer;
                  SIZE  : in integer;
                  FLAG   : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID   : in integer;
                 SHMADDR : in system.address;
                 FLAG     : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD    : in integer;
                BUFF   : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

-- Semaphore system call and C function interface
function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG  : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

end RODB_Component_Data_Types;

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

      d  b
      d  b
      d  b
r rrr      oooo      ddd d      b bbb
rr  r      o  o      d  dd      bb  b
r          o  o      d  d      b  b
r          o  o      d  d      b  b
r          o  o      d  dd      bb  b
r          oooo      ddd d      b bbb

```

```

      cccc      oooo      m m mm      p ppp
c  c      o  o      mm m m      pp  p
c          o  o      m m m      p  p
c          o  o      m m m      p  p
c  c      o  o      m m m      pp  p
cccc      oooo      m m m      p ppp
                                     p
                                     p
                                     p

```

---

Job: rodb\_component\_ada  
 Date: Tue Apr 14 22:25:57 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List  : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List  : in      RCDT.Attr_List_Type);

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end RODB_COMPONENT;

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr   r    o   o    o   o
r        o   o    o   o
r        o   o    o   o
r        o   o    o   o
r        oooo      oooo

```

```

      d   b
      d   b
      d   b
      ddd d b bbb
      d dd bb   b
      d   d b     b
      d   d b     b
      d dd bb   b
      ddd d b bbb

r rrr      oooo      ddd d
rr   r    o   o    d dd
r        o   o    d   d
r        o   o    d   d
r        o   o    d dd
r        oooo      ddd d

```

```

      cccc      oooo      m m mm      p ppp
      c   c    o   o    mm m m      pp   p
      c       o   o    m m m      p    p
      c       o   o    m m m      p    p
      c   c    o   o    m m m      pp   p
      cccc      oooo      m m m      p ppp
                                   p
                                   p
                                   p

```

---

Job: rodb\_component.ad  
Date: Tue Apr 14 22:37:48 1992

```

with TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables
Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    Flag := RCDT.READBEG(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    delay 10.0;
    Flag := RCDT.READEND(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    Flag := RCDT.WRITEBEG(Semid);

```

```

if Flag = -1 then
    raise Sem_Exception;
end if;
for I in 1.. Length loop
    if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
        raise Shm_Outrange;
    end if;
    Temp := Shmaddr + system.offset(Addr_List(I));
    if (Addr_List(I) < RCDT.CHAR_OFFSET) then
        RCDT.AINT(Temp, Attr_List(I).Int_Value);
    elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
        RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
    elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
        RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
    else
        RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
    end if;
end loop;
delay 10.0;
Flag := RCDT.WRITEEND(Semid);
if Flag = -1 then
    raise Sem_Exception;
end if;
end Write_Attrs;

```

```

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--     Number_Of_Integers
--     Position1  Integer1
--     Position2  Integer2
--     ...
--     Number_Of_Characters
--     Position1  Character1
--     Position2  Character2
--     ...
--     Number_Of_Booleans
--     Position1  Boolean1
--     Position2  Boolean2
--     ...
--     Number_Of_Floats
--     Position1  Float1
--     Position2  Float2
--     ...

```

```

procedure Load_Comps(Filename : in string) is
    Infile : FILE_TYPE;
    Temp : system.address;
    Flag : integer;
begin

```

```

    open(Infile, in_file, Filename);

```

```

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

```

```

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop

```

```

    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
end loop;
Load_Chars(Infile);

-- Initialize RODB Boolean Component
for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
end loop;
Load_Bools(Infile);

-- Initialize RODB Float Component
for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
end loop;
Load_Flts(Infile);

close(Infile);

Flag := RCDT.SEMSINIT(Semid);
if Flag = -1 then
    raise Sem_Exception;
end if;

exception
    when name_error =>
        put_line("File cannot be opened.");
        put_line("Loading components fails!");
    when data_error | end_error =>
        put_line("File format is incompatible.");
        put_line("Loading components fails!");
    when Sem_Exception =>
        put_line("Semaphore cannot be initialized.");
        raise Sem_Exception;
    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file
-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is

```



```

    Outfile : FILE_TYPE;
begin
    if Filename /= "" then
        create(Outfile, out_file, Filename,
               form=>"world=>read, owner=>read_write");
        Save_Ints(Outfile);
        Save_Chars(Outfile);
        Save_Bools(Outfile);
        Save_Flts(Outfile);
        close(Outfile);
    else
        Save_Ints(TEXT_IO.standard_output);
        Save_Chars(TEXT_IO.standard_output);
        Save_Bools(TEXT_IO.standard_output);
        Save_Flts(TEXT_IO.standard_output);
    end if;
exception
    when constraint_error =>
        put_line("RODB Components data collapsed.");
        put_line("Saving components fails!");
    when others =>
        put_line("Unknown exception.");
        put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
    Flag : integer;
begin
    Flag := RCDT.SHMDT(Shmaddr);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SEMSRMV(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Shutdown_Comps;

pragma page;

-- Load all the integers from a disk file to RODB Integer Component
procedure Load_Ints(Infile : in FILE_TYPE) is
    Length : Integer;
    Temp_Pos : integer;
    Temp_Int : integer;
    Temp_Addr : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        INT_IO.get(Infile, Temp_Int);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
            raise Shm_Outrange;
        end if;
    end loop;
end Load_Ints;

```

```

        end if;
        Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
        RCDT.AINT(Temp_Addr, Temp_Int);
    end loop;
end Load_Ints;

-- Load all the characters from a disk file to RODB Character Component
procedure Load_Chars(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Char   : character;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        get(Infile, Temp_Char);
        get(Infile, Temp_Char);           -- Skip a space
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
        RCDT.ACHAR(Temp_Addr, Temp_Char);
    end loop;
end Load_Chars;

-- Load all the booleans from a disk file to RODB Boolean Component
procedure Load_Bools(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Bool   : boolean;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        BOOL_IO.get(Infile, Temp_Bool);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
        RCDT.ABOOL(Temp_Addr, Temp_Bool);
    end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);

```

```

    FLT_IO.get(Infile, Temp_Flt);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
    RCDT.AFLT(Temp_Addr, Temp_Flt);
end loop;
end Load_Flts;

pragma page;

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

```

```

    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      eeee      aaaa      ddd d      b bbb      eeee      ggg g      cccc
rr   r    e   e      a      d   dd      bb   b      e   e      g   gg      c   c
r      eeeee      aaaaa      d   d      b   b      eeeee      g   g      c
r      e      a      a      d   d      b   b      e      e      g   g      c
r      e   e      a      aa      d   dd      bb   b      e   e      g   gg      ..  c
r      eeee      aaaa a      ddd d      b bbb      eeee      ggg g      ..  c
                                     g
                                     g
                                     gggg

```

Job: readbeg.c  
Date: Tue Apr 14 22:37:57 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[3]; /* Three semaphore operations */
    int    flag;
    void    perror();

    /* Perform three semaphore operations */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more wri
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more wri
    one_of_n_readers[2] = RREAD_START; /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 3); /* Lock the critical sec
    if (flag == -1) {
        perror("readbeg fails: ");
    }
    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d      d
      d      d
      d      d
r rrr  eeee  aaaa  ddd d  eeee  n nnn  ddd d  cccc
rr  r  e  e  a  d  dd  e  e  nn  n  d  dd  c  c
r      eeeee  aaaaa  d  d  eeeee  n  n  d  d  c
r      e  e  a  a  d  d  e  e  n  n  d  dd  c
r      e  e  a  aa  d  dd  e  e  n  n  d  dd  ..  c
r      eeee  aaaa a  ddd d  eeee  n  n  ddd d  ..  cccc

```

Job: readend.c  
 Date: Tue Apr 14 22:37:58 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int flag;
    void perror();

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);
    if (flag == -1) {
        perror("readend fails: ");
    }
    return flag;
}
/* Unlock critical section */

```



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

      i          t          b
      t          b
      t          b
w      w  r rrr      ii      ttttt      eeee      b bbb      eeee      ggg g
w  w  w  rr  r      i          t          e  e      bb  b      e  e      g  gg
w  w  w  r          i          t          eeeee      b  b      eeeee      g  g
w  w  w  r          i          t          e  e      b  b      e  e      g  g
w  w  w  r          i          t t      e  e      bb  b      e  e      g  gg
ww ww  r      iii      tt      eeee      b bbb      eeee      ggg g
                                     g
                                     g
                                     gggg
..
..

```

Job: writebeg.c  
Date: Tue Apr 14 22:38:20 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0};
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE    = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding reade
sole_writer[2] = WREAD_START;        /* preventing succeeding write
sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }
    pause();
    return flag;
}

```

```

      t
      t
r rrr .  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

                                     d
                                     d
                                     d
w      w  r rrr      ii  t      eeee  eeee  n nnn  ddd d
w w w rr  r      i  t      e  e  e  e  nn  n  d  dd
w w w r      i  t      eeeee eeeee n  n  d  d
w w w r      i  t      e  e  e  n  n  d  d
w w w r      i  t  t  e  e  e  n  n  d  dd  ..
  ww ww r      iii  tt  eeee  eeee  n  n  ddd d  ..

```

Job: writeend.c  
Date: Tue Apr 14 22:38:35 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }
    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          oooo      oooo      tt

```

```

      i          i          t
      i          i          t
      ii         nn         ttttt
      i          nn         t
      i          n         t
      i          n         t
      i          n         t  t
      iii        n         tt      ..

```

Job: semsinit.c  
 Date: Tue Apr 14 22:38:55 1992

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

ssss  eeee  m m mm  ssss  r rrr  m m mm  v      v      cccc
s    s  e    e  mm m m  s    s  rr  r  mm m m  v      v      c    c
ss      eeeee  m m m m  ss      r      m m m m  v      v      c
ss      e      m m m m  ss      r      m m m m  v      v      c
s    s  e    e  m m m m  s    s  r      m m m m  v v      ..  c    c
ssss  eeee  m m m m  ssss  r      m m m m  v      ..  cccc

```

Job: semsrmv.c  
Date: Tue Apr 14 22:39:02 1992

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semrmv fails: ");
    }
    return(flag);
}
```



```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r       o   o    o   o    t
r       o   o    o   o    t
r       o   o    o   o    t t
r       oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb      cccc      oooo      m m mm      p ppp
rr   r    o   o    d   dd   bb   b   c   c   o   o    mm m m      pp   p
r       o   o    d   d   b       b   c       o   o    m m m      p   p
r       o   o    d   d   b       b   c       o   o    m m m      p   p
r       o   o    d   dd   bb   b   c   c   o   o    m m m      pp   p
r       oooo      ddd d   b bbb      cccc      oooo      m m m      p ppp
                                     p
                                     p
                                     p

```

Job: rodbcomp.dat  
 Date: Tue Apr 14 22:41:09 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0

## **Appendix D-2**

### **Concurrent (Competing) Reader and Writer Performance Test**

**No Prevention of Preemption but with Semaphore Protection of RODB Component.**



```

      t
      t
      ttttt
      t
      t
      t
      t  t
      tt

r rrr      oooo      oooo
rr   r    o   o    o   o
r        o   o    o   o
r        o   o    o   o
r        o   o    o   o
r        oooo      oooo

```

```

      d
      d
      d
      d
      ddd d
      d  dd
      d  d
      d  d
      d  dd
      ddd d

r rrr      eeee      aaaa      ddd d
rr   r    e   e    a   a    d  dd
r        eeeeee   aaaaa   d  d
r        e       a   a    d  d
r        e   e    a   aa   d  dd
r        eeee     aaaa a   ddd d

```

```

m m mm      eeee
mm m m      e   e
m m m      eeeeee
m m m      e
m m m      e   e
m m m      eeee

```

Job: read.me  
Date: Mon Apr 20 17:46:59 1992

THERE ARE THREE TASKS RUNNING IN THE SYSTEM: TWO READERS AND ONE WRITER. THEY ARE ACCESSING THE RODB COMPONENT WHICH IS PROTECTED BY A MECHANISM IN RODBTEST1, ALL THE TASKS HAVE THE SAME PRIORITIES. THE RESULTS ARE IN FILES RODBCOMP1.OUT (RODBCOMP11.OUT AND RODBCOMP12.OUT) for 5000 reads, 5000 writes as well as 10000 reads and 5000 writes respectively. THESE CORRESPOND TO INPUT FILES RODBCOMP1.IN (RODBCOMP11.IN AND RODBCOMP12.IN

This directory stores all the files to build up RODB "attribute" component. The protection mechanism is that locking is set at the RODB level. During lock setting, there is NO prevention of preemption. Inside the C function the fast\_setprio system calls are commented out. There is only one set of UNIX semaphores in the whole system. Before actual reading, a set of five semaphore operations are imposed on the semaphores. After a read, one semaphore operation is imposed on a semaphore (decreasing reader count). Before writing there are two levels of semaphore operations which are imposed; namely, write-desire and write-lock. For write-desire only one semaphore operation is imposed on its semaphore (test&lock) and when that set and the last reader has finished, then, write-lock is set as one of a set of four semaphore operations imposed on the semaphores. After the writer finishes writing, a set of two semaphore operations are imposed on the semaphores (unlocking the RODB to allow readers or other writer in). THE INTENT IS TO DETERMINE THE EFFECT OF NOT USING PRIORITY RAISE/LOWER MANIPULATION WHICH FORMERLY GUARANTEED THE ATOMIC NATURE OF THE SEMAPHORE OPERATIONS. THIS TEST IS WITHOUT THAT COST.

## **Appendix D-3**

### **Simulated RODB**

#### **Concurrent (Competing), Reads and Writes**

**Prevention of Preemption Disabled**  
**Semaphore Protection Disabled**





```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr   oooo   oooo
rr  r   o   o   o   o
r       o   o   o   o
r       o   o   o   o
r       o   o   o   o
r       oooo   oooo

```

```

      d
      d
      d
      ddd d
      d dd
      d d
      d d
      d dd
      ddd d

r rrr   eeee   aaaa   ddd d
rr  r   e   e   a   d dd
r       eeeee   aaaaa   d d
r       e   a   a   d d
r       e   e   a   aa d dd
r       eeee   aaaa a   ddd d

```

```

m m mm   eeee
mm m m   e   e
m m m   eeeee
m m m   e
m m m   e   e
m m m   eeee

```

Job: read.me  
Date: Wed Apr 15 19:28:49 1992

THERE ARE THREE TASKS RUNNING IN THE SYSTEM: TWO READERS AND ONE WRITER. THEY ARE ACCESSING TO THE RODB COMPONENT WHICH IS NOT NOW PROTECTED BY A MECHANISM. IN Rodbtstf1 ALL THE TASKS HAVE THE SAME PRIORITY. THE RESULTS ARE IN FILES RODBCOMP1.OUT(RODBCOMP11.OUT AND RODBCOMP12.OUT). THE CORRESPONDING INPUT FILES ARE FILES RODBCOMP1.IN (RODBCOMP11.IN AND RODBCOMP12.IN).

This directory stores all the files to build up a RODB "attribute" component. The protection mechanism of locking at the RODB level is disabled. The prevention of preemption to protect the locking semaphores is also disabled. All of the protection was formerly done inside four C functions which used the fast\_setprio system call and the semop system call. There one set of UNIX semaphores formerly used in the whole system. Now, all these systems are disabled and no reader/writer protection is provided. While this would contribute to corrupt data items being read, the test was performed to see the time it would take for competing reads and writes in the "raw". Of course now the overhead of calling the read and write beginning and ending functions is of little use, but they were left in the system to provide a method of isolating the costs of the protection mechanisms.

## Appendix E

### Concurrent (Competing) Reads and Writes

Code for demonstrating the lack of mutual exclusion in a critical section for reading/writing to a simulated RODB component. The lack of mutual exclusion is presumably caused by the non-atomic nature of the semop system-call algorithm for an array of semaphores.



```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
r

oooo
o  o
o  o
o  o
o  o
oooo

oooo
o  o
o  o
o  o
oooo


```

```

      d
      d
      d
      ddd d
      d dd
      d d
      d d
      d dd
      ddd d

r rrr
rr  r
r
r
r
r
r

eeee
e  e
eeeeee
e  e
e  e
eeee

aaaa
a
aaaaa
a  a
a  aa
aaaa a


```

```

m m mm
mm m m
m m m
m m m
m m m
m m m

eeee
e  e
eeeeee
e
e  e
eeee

..
..


```

Job: read.me  
Date: Tue Apr 14 22:24:53 1992

This directory stores all the files to build up an RODB "attribute" compiler. The protection mechanism is that locking is set at the RODB level. During lock setting, there is NO prevention of preemption. WE ASSUME A SYSTEM CALL IS AN ATOMIC ACTION. Assume there is one set of three UNIX semaphores in the whole system. Before actual reading, a set of THREE semaphore operations will be imposed on the semaphores. After actual reading one semaphore array operation will be imposed on the semaphores. Before actual writing, there are two levels of operations:

write-intent on one level and write-lock and read-lock on the other.

For write-intent only one semaphore operation will be imposed on the semaphore and for write-lock a set of four semaphore operations will be ordinarily imposed on the semaphores (i.e. including test of read-lock semaphore, increase read-lock, set write-lock and clear write-intent semaphore).

BUT ON LYNX SEMAPHORE OPERATIONS, IF A PROCESS EXECUTING SEMOP IS PREEMPTED OR EVEN SLEEPS ON A SEMAPHORE EVENT, THERE IS NO GUARANTEE THAT SEMOP WILL RESUME AT THE VERY BEGINNING OF THE SEMOP ALGORITHM.

After actual writing, a set of two release-semaphore operations will ordinarily be imposed on the semaphores.

IN THIS PROGRAM, A PAUSE HAS BEEN INTRODUCED IN THE WRITER "CODE" SO THAT THESE TWO OPERATIONS (VIZ. WRITER LOCK AND READER LOCK) ARE NOT REVERSED. WHAT THIS TEST SHOWS IS FACT THAT IF THE READER HAS SLEPT ON THE WRITER INTENT SEMAPHORE THAT HAD BEEN ACTIVATED BY THE WRITER AND NOW HAD BEEN RELEASED, THE READER NOW IS AWAKENED, IT SHOULD, BUT WILL NOT GO BACK TO THE BEGINNING OF THE ALGORITHM TO CHECK THE STATUS OF THE WRITER LOCK SEMAPHORE. INSTEAD IT RESUMES WHERE IT LEFT OFF AND ONLY CHECKS THE WRITER INTENT SEMAPHORE WHICH HAS JUST BEEN CLEARED. HENCE, WE NOW HAVE BOTH A READER AND A WRITER IN THE CRITICAL SECTIONS AT THE SAME TIME.

To run this program --once compiled-- three terminals are required. Each terminal should log into the same account and execute rodb test7. When this is done a menu will appear for each process on its respective terminal screen. One process should choose option "4" to load the shared memory and press the enter key (i.e. <CR>). The question is then "asked" for the name of the file from which to load the contents of the shared memory. The simplest is to press "Enter" since there is a default data file which will be loaded by pressing the Enter key (i.e. <CR>). Once this is done the memory may be viewed by selecting option 3. Then, when the menu appears again, this first terminal should choose the option "1" to read a list of attributes followed by <CR>. When the question is asked "how many" the simplest answer is to type 1 <CR>. The question will then be raised as to the address to be "read", the simplest answer is to type the number "0" (without the quotes) BUT DO NOT PRESS "Enter". Then set up the next terminal by running another copy of rodb\_test7 and choose the option "2" to write a list of attributes <CR>. Then, when the question is "asked" for "how many", the answer is to type 1 <CR>. As before the address "question" will be asked. Type in the number "0" as before followed by <CR>. The question will then be asked for the value to be written. You may type some integer such as 200 (the number 100 is the default at address 0). Again DO NOT TYPE <CR> or Enter !! Then start the third process from the THIRD terminal. This time choose the option "1" again to read. Again select one (1) attribute and again select address "0" BUT DO NOT TYPE <CR>. NOW, go back to terminal one (1) and press Enter (<CR>). Right after that (less than five seconds) press Enter (<CR>) on terminal two (2). After that (less than five but more than two seconds) press "Enter" on terminal three (3). Clearly the first process will encounter the

semaphores before process two and will lock out process two since it will increase the reader semaphore. Process two (2) will set the write-intent semaphore but will block on the reader semaphore. Process three (3) will test the write-lock semaphore and find it NOT set (Process one (1) is structured to stay in the "reader" critical section about ten seconds so the writer will be "locked out". HOWEVER, Process three (3) will find the write-intent semaphore set so it will "sleep" on the event that the write-intent semaphore is "cleared". Meanwhile eventually -- after ten seconds -- the first reader finishes and "clears" the reader semaphore. Immediately the the second process (2) will wake to find the reader semaphore to be zero. This writer process (2) will then set the "write-lock" semaphore and will release the write-intent semaphore so as to allow higher priority readers to wake up and sleep on the "write-lock" semaphore rather than the "write-intent" semaphore (2) then enters the critical section to access the RODB Component. THIS IS WHERE THE PROBLEM APPEARS since process three (3) apparently does wake up BUT DOES NOT "GO TO START" and recheck the "write\_lock" semaphore and the "write-intent" semaphore but rather just continues on by (perhaps) not checking the "write-lock" semaphore but only the "write-intent" semaphore and finding it "clear" AND then it increases the "read-lock" semaphore and reads the shared memory. It is thus reading the shared memory at essentially shared memory. You will see the process three write out what it has read and terminate. Process two (2), which started before process three (3) is blocked at its "pause" statement just before accessing the shared memory. Process three (3) finishes even though it should be blocked from entering by the "write\_lock" semaphore set by process two (2).

Hence the Lynx semop algorithm does not meet the System V requirements and further even if it did in the non-preemptive case, it might fail under the real-time preemptive case. That is to say, if the process does not sleep but is preempted during the execution of algorithm semop when it resumed it would not know that it had been preempted and would continue on. Thus the same scenario could happen as above even if the reader process did not sleep but was preempted by an high-priority writer just before it increased the "read-lock" semaphore.

```

      r rrr      oooo      oooo      t
      rr   r    o   o      o   o    t t t t t
      r          o   o      o   o    t
      r          o   o      o   o    t
      r          o   o      o   o    t t
      r          oooo      oooo      tt

```

```

      r rrr      oooo      ddd d      d b
      rr   r    o   o      d dd      d b
      r          o   o      d d       b b b b
      r          o   o      d d       b   b
      r          o   o      d dd      b   b
      r          oooo      ddd d      b b b b

```

```

      t
      t
      t t t t t      eeee      s s s s
      t          e   e      s   s
      t          e e e e e      s s
      t          e       e      s   s
      t t        e       e      s   s
      tt        eeee      s s s s

```

---

Job: rodb\_test7.ada  
 Date: Tue Apr 14 22:25:16 1992



```

-----
--| This is main program for Test of Reader-Writer mutex problem for the RODB
-- Component. The RODB Component is represented by a small Shared Memory
-- Segment which is set up by functions and procedures in package RODBCP which
-- is a "rename" for package RODB_COMPONENT_DATA_TYPES. Most of these RODBCP
-- functions are Ada names for C library functions and UNIX System Calls.RODBC
-- operations and the accessing of the RODB Shared Memory segment.

```

```

use TEXT_IO, SYSTEM, RODB_Component_Data_Types, RODB_Component;

```

```

procedure Rodb_Test7 is

```

```

    ATTR_SIZE : constant integer := 20;
    package RCDT renames RODB_Component_Data_Types;
    package RODBCP renames RODB_Component;
    Length : integer;
    Filename : string(1..13);
    File_Len : integer;
    Addr_List : RCDT.Pos_List_Type(1..ATTR_SIZE);
    Attr_List : RCDT.Attr_List_Type(1..ATTR_SIZE);
    Choice : integer;

```

```

-- Input a list of addresses at the unit of bytes

```

```

procedure Input_Addr_List(Addr_List : in out RCDT.Pos_List_Type;
                          Length      : in      integer) is

```

```

begin

```

```

    for I in 1..Length loop
        put("Address number ");
        INT_IO.put(I, width=>3);
        put(":");
        INT_IO.get(Addr_List(I));
    end loop;
end Input_Addr_List;

```

```

-- Input a list of attributes according to their addresses

```

```

procedure Input_Attr_List(Attr_List : in out RCDT.Attr_List_Type;
                          Length      : in      integer;
                          Addr_List   : in      RCDT.Pos_List_Type) is

```

```

    An_Int : integer;
    A_Char : character;
    A_Bool : boolean;
    A_Flt  : float;
begin
    for I in 1..Length loop
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            put("Enter an attribute integer: ");
            INT_IO.get(An_Int);
            Attr_List(I) := (Type_ID => 0, Int_Value => An_Int);
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            put("Enter an attribute character: ");
            get(A_Char);
            Attr_List(I) := (Type_ID => 1, Char_Value => A_Char);
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            put("Enter an attribute boolean: ");
            BOOL_IO.get(A_Bool);
            Attr_List(I) := (Type_ID => 2, Bool_Value => A_Bool);
        else
            put("Enter an attribute float: ");
            FLT_IO.get(A_Flt);
            Attr_List(I) := (Type_ID => 3, Flt_Value => A_Flt);
        end if;
    end loop;
end Input_Attr_List;

```

```
-- Output a list of Attributes according to their addresses
procedure Output_Attr_List(Attr_List : in      RCDT.Attr_List_Type;
                           Length      : in      integer) is
```

```
begin
  for I in 1.. Length loop
    put("Attribute number");
    INT_IO.put(I, width => 3);
    put(" is ");
    case Attr_List(I).Type_ID is
      when 0 =>
        put("Integer: ");
        INT_IO.put(Attr_List(I).Int_Value);
      when 1 =>
        put("Character: ");
        put(Attr_List(I).Char_Value);
      when 2 =>
        put("Boolean: ");
        BOOL_IO.put(Attr_List(I).Bool_Value);
      when 3 =>
        put("Float: ");
        FLT_IO.put(Attr_List(I).Flt_Value);
      when others =>
        null;
    end case;
    new_line;
  end loop;
end Output_Attr_List;
```

```
begin
  loop
    put_line("1----Read a list of attributes");
    put_line("2----Write a list of attributes");
    put_line("3----print out the shared memory");
    put_line("4----Load the shared memory");
    put_line("0----Exit");
    put("Input your selection: ");
    INT_IO.get(Choice);
    skip_line;
    case Choice is
      when 0 =>
        exit;
      when 1 =>
        put("How many attributes do you want: ");
        INT_IO.get(Length);
        Input_Addr_List(Addr_List, Length);
        RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
        Output_Attr_List(Attr_List, Length);
      when 2 =>
        put("How many attributes do you want: ");
        INT_IO.get(Length);
        Input_Addr_List(Addr_List, Length);
        Input_Attr_List(Attr_List, Length, Addr_List);
        RODBCP.Write_Attrs(Addr_List, Length, Attr_List);
      when 3 =>
        put("Enter the filename to send to(none to screen): ");
        get_line(Filename, File_Len);
        RODBCP.Save_Comps(Filename(1..File_Len));
      when 4 =>
        put("Enter the filename to load from(none from rodbcomp.dat): ");
```

```

    get_line(Filename, File_Len);
    if (File_Len /= 0) then
        RODBCP.Load_Comps(Filename(1..File_Len));
    else
        RODBCP.Load_Comps("rodbcomp.dat");
    end if;
    when others =>
        put_line("Input error!");
    end case;
end loop;
RODBCP.Shutdown_Comps;
exception
    when Shm_Exception =>
        put_line("Shared memory not accessible.");
    when Shm_Outrange =>
        put_line("Shared memory out of range.");
    when Sem_Exception =>
        put_line("Semaphores not accessible.");
end Rodb_Test7;

```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
r	oooo	ddd d	b bbb

cccc	oooo	m m mm
c c	o o	mm m m
c	o o	m m m
c	o o	m m m
c c	o o	m m m
cccc	oooo	m m m

---

Job: rodb\_component\_data\_types\_  
Date: Tue Apr 14 22:25:41 1992

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY : in integer;
                 SIZE : in integer;
                 FLAG : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID : in integer;
                SHMADDR : in system.address;
                FLAG : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

```

-- Semaphore system call and C function interface
function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

```

end RODB_Component_Data_Types;

```

```

      t
      t
r rrr-  oooo  oooo  ttttt
rr  r   o  o   o  o   t
r       o  o   o  o   t
r       o  o   o  o   t
r       o  o   o  t  t
r       oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr-  oooo  ddd d  b bbb
rr  r   o  o   d  dd  bb  b
r       o  o   d  d   b   b
r       o  o   d  d   b   b
r       o  o   d  dd  bb  b
r       oooo  ddd d  b bbb

```

```

      cccc  oooo  m m mm  p ppp
c  c   o  o   mm m  m  pp  p
c  c   o  o   m  m  m  p  p
c  c   o  o   m  m  m  p  p
c  c   o  o   m  m  m  pp  p
      cccc  oooo  m  m  m  p ppp
                               p
                               p
                               p

```

Job: rodb\_component\_.ada  
Date: Tue Apr 14 22:25:57 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception  : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in      RCDT.Attr_List_Type);

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end Rodb_Component;

```



r rrr	oooo	oooo	t t tttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
r	oooo	ddd d	b bbb

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp p
c	o o	m m m	p p
c c	o o	m m m	pp p
cccc	oooo	m m m	p ppp
			p
			p
			p

Job: rodb\_component.ada  
Date: Tue Apr 14 22:37:48 1992

```

with TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    Flag := RCDT.READBEG(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    delay 10.0;
    Flag := RCDT.READEND(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    Flag := RCDT.WRITEBEG(Semid);

```

```

if Flag = -1 then
    raise Sem_Exception;
end if;
for I in 1.. Length loop
    if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
        raise Shm_Outrange;
    end if;
    Temp := Shmaddr + system.offset(Addr_List(I));
    if (Addr_List(I) < RCDT.CHAR_OFFSET) then
        RCDT.AINT(Temp, Attr_List(I).Int_Value);
    elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
        RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
    elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
        RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
    else
        RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
    end if;
end loop;
delay 10.0;
Flag := RCDT.WRITEEND(Semid);
if Flag = -1 then
    raise Sem_Exception;
end if;
end Write_Attrs;

```

```

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--      Number_Of_Integers
--      Position1 Integer1
--      Position2 Integer2
--      ...
--      Number_Of_Characters
--      Position1 Character1
--      Position2 Character2
--      ...
--      Number_Of_Booleans
--      Position1 Boolean1
--      Position2 Boolean2
--      ...
--      Number_Of_Floats
--      Position1 Float1
--      Position2 Float2
--      ...

```

```

procedure Load_Comps(Filename : in string) is
    Infile : FILE_TYPE;
    Temp : system.address;
    Flag : integer;
begin

```

```

    open(Infile, in_file, Filename);

```

```

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

```

```

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop

```

```

    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
end loop;
Load_Chars(Infile);

-- Initialize RODB Boolean Component
for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
end loop;
Load_Bools(Infile);

-- Initialize RODB Float Component
for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
end loop;
Load_Flts(Infile);

close(Infile);

Flag := RCDT.SEMSINIT(Semid);
if Flag = -1 then
    raise Sem_Exception;
end if;

exception
    when name_error =>
        put_line("File cannot be opened.");
        put_line("Loading components fails!");
    when data_error | end_error =>
        put_line("File format is incompatible.");
        put_line("Loading components fails!");
    when Sem_Exception =>
        put_line("Semaphore cannot be initialized.");
        raise Sem_Exception;
    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file
-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is

```

```

    Outfile : FILE_TYPE;
begin
    if Filename /= "" then
        create(Outfile, out_file, Filename,
               form=>"world=>read, owner=>read_write");
        Save_Ints(Outfile);
        Save_Chars(Outfile);
        Save_Bools(Outfile);
        Save_Flts(Outfile);
        close(Outfile);
    else
        Save_Ints(TEXT_IO.standard_output);
        Save_Chars(TEXT_IO.standard_output);
        Save_Bools(TEXT_IO.standard_output);
        Save_Flts(TEXT_IO.standard_output);
    end if;
exception
    when constraint_error =>
        put_line("RODB Components data collapsed.");
        put_line("Saving components fails!");
    when others =>
        put_line("Unknown exception.");
        put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
    Flag : integer;
begin
    Flag := RCDT.SHMDT(Shmaddr);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SEMSRMV(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Shutdown_Comps;

pragma page;

-- Load all the integers from a disk file to RODB Integer Component
procedure Load_Ints(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Int    : integer;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        INT_IO.get(Infile, Temp_Int);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
            raise Shm_Outrange;
        end if;
    end loop;
end Load_Ints;

```

```

        end if;
        Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
        RCDT.AINT(Temp_Addr, Temp_Int);
    end loop;
end Load_Ints;

-- Load all the characters from a disk file to RODB Character Component
procedure Load_Chars(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Char   : character;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        get(Infile, Temp_Char);
        get(Infile, Temp_Char);           -- Skip a space
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
        RCDT.ACHAR(Temp_Addr, Temp_Char);
    end loop;
end Load_Chars;

-- Load all the booleans from a disk file to RODB Boolean Component
procedure Load_Bools(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Bool   : boolean;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        BOOL_IO.get(Infile, Temp_Bool);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
        RCDT.ABOOL(Temp_Addr, Temp_Bool);
    end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);

```

```

    FLT_IO.get(Infile, Temp_Flt);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
    RCDT.AFLT(Temp_Addr, Temp_Flt);
end loop;
end Load_Flts;

pragma page;

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

```

```

    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```



```

      t
      t
r rrr.  oooo  oooo  ttttt
rr  r   o  o   o  o   t
r       o  o   o  o   t
r       o  o   o  o   t
r       o  o   o  t  t
r       oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  eeee  aaaa  ddd d  b bbb  eeee  ggg g  cccc
rr  r  e  e   a  d  dd  bb  b  e  e   g  gg  c  c
r      eeeee  aaaaa  d  d  b  b  eeeee  g  g  c
r      e      a  a  d  d  b  b  e      g  g  c
r      e  e   a  aa  d  dd  bb  b  e  e   g  gg  ..  c
r      eeee  aaaa a  ddd d  b bbb  eeee  ggg g  ..  cccc
                                g  g
                                gggg

```

Job: readbeg.c  
Date: Tue Apr 14 22:37:57 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[3]; /* Three semaphore operations */
    int    flag;
    void    perror();

    /* Perform three semaphore operations */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more wri
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more wri
    one_of_n_readers[2] = RREAD_START; /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 3); /* Lock the critical sec
    if (flag == -1) {
        perror("readbeg fails: ");
    }
    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

      d      d      d
      d      d      d
      d      d      d
r rrr      eeee      aaaa      ddd d      eeee      n nnn      ddd d      cccc
rr  r      e  e      a      d dd      e  e      nn  n      d dd      c  c
r          eeeee      aaaaa      d  d      eeeee      n  n      d  d      c
r          e  e      a  a      d  d      e  e      n  n      d  d      c
r          e  e      a  aa      d dd      e  e      n  n      d dd      .. c
r          eeee      aaaa a      ddd d      eeee      n  n      ddd d      .. cccc

```

Job: readend.c  
Date: Tue Apr 14 22:37:58 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int flag;
    void perror();

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);
    if (flag == -1) {
        perror("readend fails: ");
    }
    return flag;
}
/* Unlock critical section */

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          oooo      oooo      tt

```

```

      i      t      b
      t      b
      t      b
w      w      r rrr      ii      ttttt      eeee      b bbb      eeee      ggg g
w w w rr  r      i      t      e e      bb  b      e e      g gg
w w w r      i      t      eeeee      b  b      eeeee      g g
w w w r      i      t      e      b  b      e      g g
w w w r      i      t t      e e      bb  b      e e      g gg
ww ww r      iii      tt      eeee      b bbb      eeee      ggg g
                                     g
                                     g
                                     gggg
..
..

```

Job: writebeg.c  
Date: Tue Apr 14 22:38:20 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS    = { 0, 0, 0};
struct sembuf WREAD_START        = { 0, 1, 0};
struct sembuf WWRITE_LOCK        = { 1, 1, 0};
struct sembuf WWRITE_DESIRE      = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding read */
    sole_writer[2] = WREAD_START;      /* preventing succeeding writ
sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }
    pause();
    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          o  o      o  o      tt
r          oooo      oooo

```

```

                                     d
                                     d
                                     d
      i          t          ttttt      eeee      eeee      n nnn      ddd d
w      w      r rrr      ii          t          e   e      e   e      nn   n      d   dd
w w w      rr  r      i          t          eeeee      eeeee      n   n      d   d
w w w      r          i          t          e          e          n   n      d   d
w w w      r          i          t t       e   e      e   e      n   n      d   dd
ww ww      r          iii         tt        eeee      eeee      n   n      ddd d

```

Job: writeend.c  
Date: Tue Apr 14 22:38:35 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int      flag;
    void      perror();

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }
    return flag;
}

```



			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

				i		i	t	
							t	
ssss	eeee	m m mm	ssss	ii	n nnn	ii	ttttt	
s s	e e	mm m m	s s	i	nn n	i	t	
ss	eeeeee	m m m	ss	i	n n	i	t	
ss	e	m m m	ss	i	n n	i	t	
s s	e e	m m m	s s	i	n n	i	t t	..
ssss	eeee	m m m	ssss	iii	n n	iii	tt	..

Job: semsinit.c  
Date: Tue Apr 14 22:38:55 1992

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

ssss	eeee	m m mm	ssss	r rrr	m m mm	v	v	cccc
s s	e e	mm m m	s s	rr r	mm m m	v	v	c c
ss	eeeeee	m m m	ss	r	m m m	v	v	c
ss	e	m m m	ss	r	m m m	v	v	c
s s	e e	m m m	s s	r	m m m	v v	..	c c
ssss	eeee	m m m	ssss	r	m m m	v	..	cccc

Job: semsrmv.c  
Date: Tue Apr 14 22:39:02 1992

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

      d  b
      d  b
      d  b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr  r      o  o      d  dd      bb  b      c  c      o  o      mm m m      pp  p
r          o  o      d  d      b  b      c      o  o      m m m      p  p
r          o  o      d  dd      bb  b      c  c      o  o      m m m      pp  p
r          oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp
                                     p
                                     p
                                     p

```

Job: rodbcomp.dat  
Date: Tue Apr 14 22:41:09 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0



---

Copies of this publication have been deposited with the Texas State Library in compliance with the State Depository Law.

---



# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

## **Appendices A & B**

*Prepared for*

***NASA/JSC Data Management Systems  
for Space Station Freedom (SSF)***

Prepared by Co-Principal Investigators:

**T. F. Leibfried Jr.  
Sadeqh Davari**  
*University of Houston-Clear Lake*

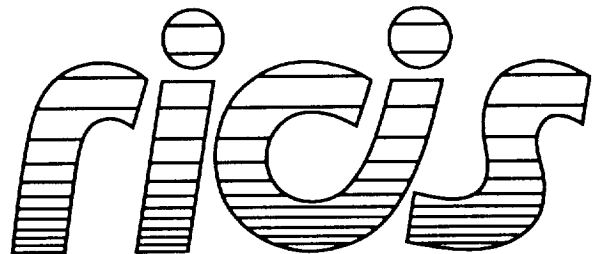
**Swami Natarajan  
Wei Zhao**  
*Texas A&M University*

Research Associates:

**Libin Wu**  
*University of Houston-Clear Lake*

**Gary Smith**  
*Texas A& M University*

April 1992



*Research Institute for Computing and Information Systems  
University of Houston-Clear Lake*

***F • I • N • A • L     R • E • P • O • R • T***

## *The RICIS Concept*

---

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

**APPENDICES  
TO  
FINAL REPORT  
VOL 1 OF 3  
APPENDICES A & B**



# **Appendix A**

## **Ada Preprocessor**



-----  
--  
-- Preprocessor Version 0.5  
--  
-----

-- Changes from version 0.4  
--

-- User enters input and output filenames.  
--

-- Ignores comments better (hopefully completely)  
--

-- List\_pack doesn't allocate storage until actually needed.  
--  
-----

-- Assumptions/limitations:  
--

-- 1. Input and Output must be different files.  
--

-- 2. Correct Syntax. (input.ada must be a compilable Ada program)  
--

-- 3. Flat name space in input.ada  
--

-- 4. The read\_open calls and all of its associated attribute\_read calls  
-- are in input.ada.  
--

-- 5. The specification and body of read\_open and attribute\_read are  
-- given in another routine (ie: 'with'ed from a package)  
--

-- 6. Each read\_open will use a distinct handle.  
--

-- 7. Each attribute\_read uses only one handle (not a list).  
--

-- 8. The handle in an attribute\_read is the exact name given in  
-- corresponding read\_open.  
--

-- 9. The syntax of the read\_open and attribute\_read is just like  
-- that in the CDR document. (except no list of handles).  
--

-- 10. This routine produces new Ada code, which must then be separately  
-- compiled.  
--

-- 11. package 'dummy\_pack' is created. (decreased name space)  
--

-- 12. A MODB routine is simulated in the preprocessor - when  
-- given the name of an attribute, returns the type of that  
-- attribute.  
--

-- 13. The first begin in the file is the begin of the main program.  
-- This was necessary since the best place to put dummy\_pack just before  
-- it. (there are ways around this - as well as several of the above)  
--  
-----  
-----

-----  
-----  
-- This is the simulated MODB package and function call  
--  
-----  
-----

package MODB\_package is  
function Get\_type (What : in String) return String;  
end MODB\_package;

```
package body MODB_package is
```

```
-----  
-- when passed an attribute name, return the corresponding type  
-----
```

```
function Get_type (What : in String) return String is  
begin
```

```
    if (What = "AP TELESCOPE.X_POSITION") then  
        return "X_POSITION T";  
    elsif (What = "AP TELESCOPE.Y_POSITION") then  
        return "Y_POSITION T";  
    elsif (What = "AP TELESCOPE.SHUTTER") then  
        return "SHUTTER_POSITION T";  
    elsif (What = "AP TELESCOPE.RIGHT_ASC") then  
        return "RIGHT_ASCENSION T";  
    elsif (What = "AP TELESCOPE.DECLINATION") then  
        return "DECLINATION T";  
    elsif (What = "AP TELESCOPE.DEC DEGREES") then  
        return "DECLINATION DEGREES T";  
    elsif (What = "AP TELESCOPE.DEC MINUTES") then  
        return "DECLINATION MINUTES T";  
    elsif (What = "AP TELESCOPE.DEC SECONDS") then  
        return "DECLINATION SECONDS T";  
    elsif (What = "AP TELESCOPE.STATE") then  
        return "STATE T";  
    else  
        return "Unknown_type";  
    end if;
```

```
end Get_type;
```

```
end MODB_package;
```

```
-----  
-- package LIST_PACK  
--
```

```
-- This is a generic list package that must be instantiated with  
-- A data type.
```

```
-- Unconstrained array type 'list_type' is made available as a  
-- private type. The 'with' should create a list of this  
-- type and pass it back as a parameter to the calls in this  
-- package. It was done this way so that several lists can be worked on  
-- by each instantiation, while still keeping the workings of the  
-- lists private.  
--
```

```
-----  
with Text_io; use Text_io;
```

```
Generic
```

```
    type Item_type is private;
```

```
package List_pack is
```

```
--  
-- This is the type that is used by 'with' to create a list.  
--
```

```
type List_type(Max: Natural) is private;
```

```
function Size of(Lister : in List_type) return Integer;
```

```
function Get_Item(Offset : Integer; Lister : in List_type) return Item_type;
```

```
procedure Add_item(Item: in Item_type; Lister : in out List_type);
```

```
procedure Update_item(Item: in Item_type; Lister : in out List_type;  
                      Offset : Natural);
```

```
private
```

```
    type Item_ptr_type is access Item_type;
```



```

type List_array is array(Integer range <>) OF Item_ptr_type;
type List_type(Max: Natural ) is
  record
    List : List_array(0..Max);
    Num_in : Integer := 0;
  end record;

end List_pack;

-----
-- Body
-----
package BODY List_pack is

  -----
  -- Add to the list
  -----
  procedure Add_item(Item: in Item_type; Lister : in out List_type) is
  begin
    if Lister.Num_in = Lister.Max then
      Put(" ERROR: List_pack.Add_item: Lister of items is full - didn't add");
    else
      Lister.List(Lister.Num_in) := new Item_type'(item);
      Lister.Num_in := Lister.Num_in + 1;
    end if;
  end Add_item;

  -----
  -- Update an Item
  -----
  procedure Update_item(Item: in Item_type; Lister : in out List_type;
    Offset : Natural) is
  begin
    if Lister.Num_in < Offset then
      Put(" ERROR: List_pack.Update_item: offset out of current range");
    else
      Lister.List(Offset - 1).all := Item;
    end if;
  end Update_item;

  -----
  -- Return (without) deleting, the item at a particular offset
  -----
  function Get_item(Offset : Integer; Lister : in List_type) return Item_type is
  begin
    if Lister.Num_in = 0 then
      Put(" ERROR: List_pack.Get item: Lister of items is empty- can't retrieve");
    elsif Offset > Lister.Num_in then
      Put(" ERROR: List_pack.Get_item: not that many items - can't retrieve");
    else
      return Lister.List(Offset - 1).all;
    end if;
  end Get_item;

  -----
  -- Returns how many items are currently in the list
  -----
  function Size_of(Lister: in List_type) return Integer is
  begin
    return Lister.Num_in;
  end;
end List_pack;

-----
-----

```

```

-- package inout_pack
--
-- This is a package to handle all reading from the input file and
-- writing to the output file
-- Upon instantiation, the input and output files are opened
-- Right now, they are static filenames
--
-----
with Text_io; use Text_io;
package Inout_pack is

    --
    -- Supply types and max lengths for the 'words' returned each
    -- call to get_word
    --
    Max_name_len : constant := 80;
    subtype Name_len_type is Integer range 0..Max_name_len;
    subtype Name_type is String(1..Max_name_len);

    --
    -- Declare a word type to be a name type and a current length
    -- Didn't make it private because of the plethora of times
    -- the individual fields are used elsewhere.
    --
    type Word_type is
        record
            Name : Name_type;
            Len : Name_len_type;
        end record;

    function Get_next_word return Word_type;
    procedure Put_f (What : in String);
    procedure Put_line_f (What : in String := "");
    procedure Reset_input;
    function Open_files(inn_fname,out_fname : string) return boolean;
end Inout_pack;

-----
package BODY Inout_pack is

    Innf : File_type;
    Outf : File_type;
    Lookahead : Character := Ascii.nul;    -- next character in the input

    -----
    -- resets the input file back to the beginning of the file
    -----
    procedure Reset_input is
    begin
        Reset(Innf);
        Lookahead := Ascii.nul;
    end;

    -----
    -- Outputs a string to the output file
    -----
    procedure Put_f (What : in String) is
    begin
        Put(Outf,What);
    end Put_f;

    -----
    -- Outputs a string and new_line to the output file
    -----

```

```

procedure Put_line_f (What : in String := "") is
begin
  Put_line(Outf,What);
end Put_line_f;

```

```

-----
-- Scans the input file. Returns the next 'word'. Entire comments
-- are passed back as words. Blanks, line feeds, etc. are passed
-- back as one character long words.
-- Passes back an ascii.nul and length of 0 on end-of-file
-- It finds a single word by stopping at blanks, etc. This stop
-- character is stored in the internal (to body) variable
-- called lookahead. Lookahead is then used as the first character
-- on the next call
-----

```

```

function Get_next_word return Word_type is

```

```

  Word : Word_type;
  Char : Character;
  Cnt : Integer;

```

```

--

```

```

-- The following to_upper was taken out of the Verdex library
--

```

```

type convert_t is array(character) of character;

```

```

to_upper: constant convert_t := (
  ascii.nul, ascii.soh, ascii.stx, ascii.etx,
  ascii.eot, ascii.enq, ascii.ack, ascii.bel,
  ascii.bs, ascii.ht, ascii.lf, ascii.vt,
  ascii.ff, ascii.cr, ascii.so, ascii.si,
  ascii.dle, ascii.dcl, ascii.dc2, ascii.dc3,
  ascii.dc4, ascii.nak, ascii.syn, ascii.etb,
  ascii.can, ascii.em, ascii.sub, ascii.esc,
  ascii.fs, ascii.gs, ascii.rs, ascii.us,
  ' ', '!', '"', '#', '$', '%', '&', '\'',
  '(', ')', '*', '+', ',', '-', '.', '/',
  '0', '1', '2', '3', '4', '5', '6', '7',
  '8', '9', ':', ';', '<', '=', '>', '?',
  '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
  'X', 'Y', 'Z', '[', '\', ']', '^', '_',
  '\', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
  'X', 'Y', 'Z', '{', '|', '}', '~', ascii.del);

```

```

-----
-- Gets the next character
-----

```

```

function Get_char return Character is

```

```

  Temp_char : Character;
begin
  if End_of_file(Innf) then
    return Ascii.nul;
  elsif End_of_line(Innf) then
    Skip_line(Innf);
    return Ascii.lf;
  else
    Get(Innf,Temp_char);
    return to_upper(Temp_char);
  end if;
end Get_char;

```

```

begin

```

```

  if (End_of_file(Innf)) AND (Lookahead = Ascii.nul) then
    Word.Len := 0;

```

```

    Lookahead := Ascii.nul;
else
    if Lookahead = Ascii.nul then
        Word.Name(1) := Get_char;
    else
        Word.Name(1) := Lookahead;
    end if;
    Word.Len := 1;

    --
    -- Found the start of a word, get rest of word
    --
    if (Lookahead in 'A' .. 'Z') then
        loop
            Char := Get_char;
            exit when (Char not in 'A'..'Z') AND
                (Char not in '1'..'9') AND (Char /= '_');
            Word.Len := Word.Len + 1;
            Word.Name(Word.Len) := Char;
        end loop;
        Lookahead := Char;

        --
        -- Look to see if it is a comment, if so get entire comment
        -- To do this, have to read next character. if it is not
        -- a minus, pass the first back this time as word(1) and
        -- store the second in Lookahead
        --
        elsif (Lookahead = '-') then
            Char := Get_char;
            if (Char = '-') then
                Word.Len := 2;
                Word.Name(Word.Len) := Char;
                get_line(Innf, Word.Name(3..Word.Name'last), Cnt);
                Word.Len := Cnt;
                Lookahead := Ascii.lf;
            else
                Lookahead := Char;
            end if;

            --
            -- else just pass that one character back as word(1) and get
            -- next character to the lookahead
            --
        else
            Lookahead := Get_char;
        end if;
    end if;

    return Word;
end Get_next_word;

function Open_files(inn_fname, out_fname : string) return boolean is
begin
    Open (Innf, in_file, inn_fname);
    Create (Outf, out_file, out_fname);
    return true;
exception
    when others => return false;
end;
end Inout_pack;

```

---



---

```

-- procedure PROCESS
--
-- This is the main preprocessor routine
--
-----
-----
with Text_io; use Text_io;
with List_pack;           -- Generic list package

with Inout_pack; use Inout_pack; -- Package to handle all I/O
                                -- includes definitions for Words, etc.
                                -- 'Used' it because it is used so often

with MODB_Package;

procedure Preprocess is

--
-- These set upper bounds on the number of read_opens, attribute_reads
-- and the number of components in an attribute or address array
--
Max_num_opens : constant Integer := 20;
Max_num_reads : constant Integer := 100;
Max_items : constant Integer := 20;
Dummy_pack_name : constant String := "dummy_pack";

--
-- Set up a list of Words so that it can be used to hold the parameters
-- to the read_opens and attribute reads
--
package Word_pack is NEW List_pack(Word_type);

--
-- Set up a list of read_open(s), each of which has a Handle name,
-- attribute list name, and a list to store the actual fields in
-- the attribute list - filled in on second pass
--
Type Open_type is record
  Handle : Word_type := ((others => ' '),0);
  Ary_id : Word_type := ((others => ' '),0);
  Ary : Word_pack.List_type(Max_items);
end record;
package Open_pack is NEW List_pack(Open_type);
Open_list : Open_pack.List_type(Max_num_opens);

--
-- Set up a list of attribute_read(s), each of which has an
-- address list name and a list to store the actual fields in
-- the address list - filled in on second pass
--
Type Read_type is record
  Ary_id : Word_type := ((others => ' '),0);
  Ary : Word_pack.List_type(Max_items);
end record;
package Read_pack is NEW List_pack(Read_type);
Read_list : Read_pack.List_type(Max_num_reads);

-----
-- Search through the list of attribute_reads for an address parameter
-- that is equal to Word. Return its Index if found, zero otherwise
-----
function Srch_read Ary_id(Word : Word_type) return Integer is
  Temp_read : Read_type;
begin

  FOR Index in 1..Read_pack.Size_of(Read_list) loop

```

```

    Temp_read := Read_pack.Get_item(Index, Read_list);
    if (Temp_read.Ary_id.Len = Word.Len)
        and (Temp_read.Ary_id.Name(1..Word.Len)
            = Word.Name(1..Word.Len)) then
        return Index;
    end if;
end loop;
return 0;
end;

```

```

-----
-- Search through the list of read opens for an attribute parameter
-- that is equal to Word. Return its Index if found, zero otherwise
-----

```

```

function Srch_open_Ary_id(Word:Word_type) return Integer is
    Temp_open : Open_type;
begin

```

```

    FOR Index in 1..Open_pack.Size_of(Open_list) loop
        Temp_open := Open_pack.Get_item(Index,Open_list);
        if (Temp_open.Ary_id.Len = Word.Len)
            and (Temp_open.Ary_id.Name(1..Word.Len)
                = Word.Name(1..Word.Len)) then
            return Index;
        end if;
    end loop;
    return 0;
end;

```

```

-----
-- Search through the list of read opens for a Handle parameter
-- that is equal to Word. Return its Index if found, zero otherwise
-----

```

```

function Srch_open_Handle(Word:Word_type) return Integer is
    Temp_open : Open_type;
begin

```

```

    FOR Index in 1..Open_pack.Size_of(Open_list) loop
        Temp_open := Open_pack.Get_item(Index,Open_list);
        if (Temp_open.Handle.Len = Word.Len)
            and (Temp_open.Handle.Name(1..Word.Len)
                = Word.Name(1..Word.Len)) then
            return Index;
        end if;
    end loop;
    return 0;
end;

```

```

-----
-- Outputs the complete dummy package with procedures for each valid
-- Handle along with the appropriate formal parameters
-----

```

```

procedure Write_package is
    Param_name : String(1..1);
    T_open : Open_type;
    T_read : Read_type;
    T_word : Word_type;
begin

```

```

    --
    -- output the package specification
    -- For each read_open, output a procedure spec for it using the
    -- Handle as the name of the procedure along with an extension
    -- include appropriate parameters for each field in its
    -- attribute list, starting with 'a' - name doesn't matter
    --

```

```

Put_line_f("-- THIS is A DUMMY package ADDED BY THE PREPROCESSOR --");
Put_line_f("package " & Dummy_pack_name & " is");

FOR i in 1..Open_pack.Size_of(Open_list) loop
  T_open := Open_pack.Get_item(i,Open_list);
  T_word := T_open.Handle;
  Put_line_f("      procedure " & T_word.Name(1..T_word.Len) &
                                                    "_dummy (");

  Param_name := "a";
  T_word := Word_pack.Get_item(1,T_open.Ary);
  Put_f("      " & Param_name & " : " &
        T_word.Name(1..T_word.Len));

  FOR k in 2..Word_pack.Size_of(T_open.Ary) loop
    Param_name(1) := character'succ(param_name(1));
    Put_line_f(";");
    T_word := Word_pack.Get_item(k,T_open.Ary);
    Put_f("      " & Param_name & " : " &
          T_word.Name(1..T_word.Len));

  end loop;
  Put_line_f(";");
end loop;
Put_line_f("end " & Dummy_pack_name & ";");

--
-- Output the Package body
-- For each read_open, output a Procedure body for it using the
-- Handle as the name of the procedure along with an extension
-- include appropriate parameters for each field in its
-- attribute list, starting with 'a' - name doesn't matter
-- only Statement in body should be null
--
Put_line_f("package body " & Dummy_pack_name & " is");
FOR i in 1..Open_pack.Size_of(Open_list) loop
  T_open := Open_pack.Get_item(i,Open_list);
  T_word := T_open.Handle;
  Put_line_f("      procedure " & T_word.Name(1..T_word.Len) &
                                                    "_dummy (");

  Param_name := "a";
  T_word := Word_pack.Get_item(1,T_open.Ary);
  Put_f("      " & Param_name & " : " &
        T_word.Name(1..T_word.Len));

  FOR k in 2..Word_pack.Size_of(T_open.Ary) loop
    Param_name(1) := character'succ(param_name(1));
    Put_line_f(";");
    T_word := Word_pack.Get_item(k,T_open.Ary);
    Put_f("      " & Param_name & " : " &
          T_word.Name(1..T_word.Len));

  end loop;
  Put_line_f("      is ");
  Put_line_f("      begin");
  Put_line_f("      null;");
  Put_line_f("      end;");
end loop;
Put_line_f("end " & Dummy_pack_name & ";");
Put_line_f;

end;

-----
-- Outputs a call to the appropriate dummy procedure from the read_opens
-- uses Handle with extension as the procedure name and includes
-- actual paramters for each of the fields in the address list
-----
procedure Write_call(Handle : in Word_type;addr:in Word_type) is
  T_read : Read_type;
  T_word : Word_type;
  Index : Integer;

```

```

begin
  Put_line_f;
  Put_f("---- This is a call to the dummy procedure in dummy ");
  Put_line_f("package ----");
  Put_line_f(Dummy_pack_name & "." & Handle.Name(1..handle.Len) &
                                                    "_dummy (");
  Index := Srch_read Ary_id(addr);
  T_read := Read_pack.Get_item(Index,Read_list);
  T_word := Word_pack.Get_item(1,T_read.Ary);
  Put_f("      " & T_word.Name(1..T_word.Len));
  FOR i in 2..Word_pack.Size_of(T_read.Ary) loop
    Put_line_f(",");
    T_word := Word_pack.Get_item(i,T_read.Ary);
    Put_f("      " & T_word.Name(1..T_word.Len));
  end loop;
  Put_line_f(";");
end;

```

```

-----
-- Scan the input for attribute_read and read_open
-- Get the names of the attribute and address_arrays
-----

```

```

procedure Pass_one (Open_list : in out Open_pack.List_type;
                    Read_list : in out Read_pack.List_type) is

```

```

  Word : Word_type := ((others => ' '),0);
  T_open : Open_type;
  T_read : Read_type;
  State: Integer range 0..4 := 0;

```

```

begin
  loop
    Word := Get_next_word;
    exit when Word.Len = 0;
    if (Word.Name(1) /= ' ') and
        (Word.Name(1) /= Ascii.ht) and
        (Word.Name(1) /= Ascii.lf) and
        (not ((Word.Len > 2) and then
              ((Word.Name(1) = '-') and (Word.Name(2) = '-')))) then
      case State is
        when 0 =>
          if (Word.Name(1..Word.Len) = "READ_OPEN") then
            State := 1;
            T_open.Ary_id := ((others => ' '),0);
            T_open.Handle := ((others => ' '),0);
          elsif (Word.Name(1..Word.Len) = "ATTRIBUTE_READ") then
            State := 3;
            T_read.Ary_id := ((others => ' '),0);
          end if;
        when 1 =>
          if (Word.Name(1) /= ',') then
            T_open.Ary_id.Name(1..Word.Len) := Word.Name(1..Word.Len);
            T_open.Ary_id.Len := Word.Len;
          else
            State := 2;
          end if;
        when 2 =>
          if (Word.Name(1) /= ',') then
            T_open.Handle.Name(1..Word.Len) := Word.Name(1..Word.Len);
            T_open.Handle.Len := Word.Len;
          else
            if Srch_open Handle(Word) = 0 then
              Open_pack.Add_item(T_open,Open_list);
            else
              Put_line("Error: same Handle used in two read_opens");
            end if;
          end if;
        -- State 3 and 4 are not reached in this version
      end case;
    end if;
  end loop;
end;

```



```

        State := 0;
    end if;
when 3 =>
    if (Word.Name(1) = ',') then
        State := 4;
    end if;
when 4 =>
    if (Word.Name(1) /= ',') then
        T_read.Ary_id.Name(1..Word.Len) := Word.Name(1..Word.Len);
        T_read.Ary_id.Len := Word.Len;
    else
        if Srch_read_Ary_id(Word) = 0 then
            Read_pack.Add_item(T_read, Read_list);
        end if;
        State := 0;
    end if;
end case;
end if;
end loop;
end Pass_one;

```

```

-----
-- Scan the input for the delcarations of attribute and address arrays
-- Store the fields in corresponding list
-----

```

```

procedure Pass_two (Open_list : in out Open_pack.List_type;
                    Read_list : in out Read_pack.List_type) is
    Word : Word_type := (others => ' '), 0;
    T_word : Word_type := (others => ' '), 0;
    Attr_type word : Word_type := (others => ' '), 0;
    T_open : Open_type;
    T_read : Read_type;
    Index : Integer;
    State: Integer range 0..9 := 0;

begin
    Reset_input;
    State := 0;
    loop
        Word := Get_next_word;
        exit when Word.Len = 0;
        if (Word.Name(1) /= ' ') and
            (Word.Name(1) /= Ascii.ht) and
            (Word.Name(1) /= Ascii.lf) and
            (not ((Word.len > 2) and then
                ((Word.Name(1) = '-') and (Word.Name(2) = '-')))) then
            case State is
                when 0 =>
                    if (Srch_open_Ary_id(Word) > 0) then
                        State := 1;
                        Index := Srch_open_Ary_id(Word);
                        T_open := Open_pack.get_item(Index, Open_list);
                    elsif (Srch_read_Ary_id(Word) > 0) then
                        State := 6;
                        Index := Srch_read_Ary_id(Word);
                        T_read := Read_pack.get_item(Index, Read_list);
                    end if;
                when 1 =>
                    if (Word.Name(1) = ':') then
                        State := 2;
                    elsif (Word.Name(1) = ',') then
                        State := 0;
                    end if;
                when 2 =>
                    if (Word.Name(1) = '(') then
                        State := 3;

```

```

        end if;
    when 3 =>
        if (Word.Name(1) = '(') then
            State := 4;
            T_word.Len := 0;
        end if;
    when 4 =>
        if (Word.Name(1) = ',') then
            T_word.Name(T_word.Len+1) := '.';
            T_word.Len := T_word.Len + 1;
        elsif (Word.Name(1) /= '(') then
            T_word.Name(T_word.Len+1..T_word.Len+Word.Len) :=
                Word.Name(1..Word.Len);
            T_word.Len := T_word.Len + Word.Len;
        else
            declare
                Attr_type : constant String :=
                    MODB_package.Get_type(T_word.name(1..T_word.Len));
            begin
                Attr_type_word.name(1..Attr_type'length) := Attr_type;
                Attr_type_word.Len := Attr_type'length;
            end;
            Word_pack.Add_item(Attr_type_word, T_open.Ary);
            Open_pack.Update_item(T_open, Open_list, Index);
            State := 5;
        end if;
    when 5 =>
        if (Word.Name(1) = ',') then
            State := 3;
        elsif (Word.Name(1) = ')') then
            State := 0;
        end if;
    when 6 =>
        if (Word.Name(1) = ':') then
            State := 7;
        elsif (Word.Name(1) = ',') then
            State := 0;
        end if;
    when 7 =>
        if (Word.Name(1) = '(') then
            State := 8;
            T_word.Len := 0;
        end if;
    when 8 =>
        if (Word.Name(1) /= '') then
            T_word.Name(T_word.Len+1..T_word.Len+Word.Len) :=
                Word.Name(1..Word.Len);
            T_word.Len := T_word.Len + Word.Len;
        else
            Word_pack.Add_item(T_word, T_read.Ary);
            Read_pack.Update_item(T_read, Read_list, Index);
            State := 9;
        end if;
    when 9 =>
        if (Word.Name(1) = ',') then
            T_word.Len := 0;
            State := 8;
        elsif (Word.Name(1) = ')') then
            State := 0;
        end if;

    end case;
end if;
end loop;
end Pass_two;

```

```

-----
-- Echo the input to the output along with the dummy package and
-- dummy procedure calls
-----

```

```

procedure Pass_three (Open_list : in out Open_pack.List_type;
  Read_list : in out Read_pack.List_type) is
  Word : Word_type := ((others => ' '),0);
  T_Handle : Word_type := ((others => ' '),0);
  T_addr : Word_type := ((others => ' '),0);
  State: Integer range 0..5 := 0;

begin
  Reset_input;
  loop
    Word := Get_next_word;
    exit when Word.Len = 0;
    case State is
      when 0 =>
        if (Word.Name(1..Word.Len) = "BEGIN") then
          Write_package;
          State := 1;
        end if;
      when 1 =>
        if (Word.Name(1..Word.Len) = "ATTRIBUTE_READ") then
          State := 2;
        end if;
      when 2 =>
        if (Word.Name(1) /= ',') then
          T_Handle.Name(1..Word.Len) := Word.Name(1..Word.Len);
          T_Handle.Len := Word.Len;
        else
          State := 3;
        end if;
      when 3 =>
        if (Word.Name(1) /= ',') then
          T_addr.Name(1..Word.Len) := Word.Name(1..Word.Len);
          T_addr.Len := Word.Len;
        else
          State := 4;
        end if;
      when 4 =>
        if (Word.Name(1) = ';') then
          State := 5;
        end if;
      when 5 =>
        Write_call(T_handle,T_addr);
        State := 1;
    end case;
    if (word.name(1) = ascii.lf) then
      Put_line_f;
    else
      Put_f(Word.Name(1..Word.Len));
    end if;
  end loop;
  Put_line_f; -- forces a new_line at end of output file
end Pass_three;

```

```

-----
-- Main program
--

```

```

-- It works in three main passes:
-- 1. Scans the input and find each occurrence of 'open_read' and
--    'read_open'.
--    For each 'open_read', adds an 'open' item to the open_list
--    and stores the handle and name of the attribute array in that

```

```

--      item.
--      For each 'attribute_read', adds a 'read' item to the read_list
--      and stores the name of the address array in that item.
--
--      2. Scans the input and find where each of the attribute arrays
--      and address arrays are declared.
--      For each attribute array found, update that item in the open list:
--      For each component in the declaration, add an entry in the
--      'ary' part of the open item that contains the type of that
--      component.
--      For each address array found, update that item in the read list:
--      For each component in the declaration, add an entry in the
--      'ary' part of the read item that contains the name of the
--      component without the address qualifier.
--
--      3. Just before the begin of the main procedure, output a dummy
--      package that contains a procedure for each of the handles
--      in the open_list. The parameters to each array will have
--      dummy names but will be of the types obtained in the
--      attribute array.
--      For each of 'attribute_read': immediately after, place a call to
--      the appropriate dummy procedure. The parameters will be the
--      components found in the address array, except without the
--      address qualifier.
--
-----
begin
-----
--  Open input and Output files
-----
  declare
    inn_fname : string(1..Max_name_len);
    inn_size : integer;
    out_fname : string(1..Max_name_len);
    out_size : integer;
  begin
    loop
      put("Enter input file name -> ");
      get_line(inn_fname,inn_size);
      put("Enter output file name (not same as input) -> ");
      get_line(out_fname,out_size);
      exit when ((inn_size /= out_size) or else
        (inn_fname(1..inn_size) /= out_fname(1..out_size)));
      put_line("Input and Output must be different files");
    end loop;

    if Open_files(inn_fname(1..inn_size),out_fname(1..out_size))
    then
      Pass_one(Open_list,Read_list);
      Pass_two(Open_list,Read_list);
      Pass_three(Open_list,Read_list);
    else
      put_line("Error opening files");
    end if;
  end;
end Preprocess;

```

## **Appendix B**

### **IPC Message Queue Performance**



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d		
			d		
			d		
r rrr	eeee	aaaa	ddd d	m m mm	eeee
rr r	e e	a	d dd	mm m m	e e
r	eeeeee	aaaaa	d d	m m m	eeeeee
r	e	a a	d d	m m m	e
r	e e	a aa	d dd	..	m m m e e
r	eeee	aaaa a	ddd d	..	m m m eeee

Job: read.me  
Date: Sat Mar 28 21:21:56 1992

This directory stores all the files to simulate the IPC MESSAGE QUEUE COMMUNICATION between processes. Two processes are created with one program using the fork system call. One is the sender (parent) and the other is the receiver (child). The sender sends a short message (request) to the receiver through one IPC Message Queue. The receiver receives the request and sends a response back to the sender through another IPC Message Queue. The sender receives the response message. This terminates one IPC communication. TESTING IS DONE FOR THE NUMBER OF SENDING AND RECEIVING CYCLES FROM 1000 to 5000 in steps of 1000. The round-trip time measured results are in file rodbcomp1.out for the receiver and in file rodbcomp2.out for the sender.



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b					
		d	b					t
		d	b					t
r rrr	oooo	ddd d	b bbb		t	eeee	ssss	ttttt
rr r	o o	d dd	bb b		t	e e	s s	t
r	o o	d d	b b		t	eeeeee	ss	t
r	o o	d d	b b		t	e	ss	t
r	o o	d dd	bb b		t t	e e	s s	t t
r	oooo	ddd d	b bbb		tt	eeee	ssss	tt

```
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_Test_Data is
```

```
    -- Message queue system call interface
    function MSGGET(KEY : in integer;
                    FLAG : in integer) return integer;
    pragma INTERFACE(C, MSGGET);
    pragma INTERFACE NAME(MSGGET, "msgget");
    function MSGSND(MSGID : in integer;
                    MSGP : in SYSTEM.address;
                    MSGSZ : in integer;
                    MSGFLG : in integer) return integer;
    pragma INTERFACE(C, MSGSND);
    pragma INTERFACE NAME(MSGSND, "msgsnd");
    function MSGRCV(MSGID : in integer;
                    MSGP : in SYSTEM.address;
                    MSGSZ : in integer;
                    MSGTYP : in integer;
                    MSGFLG : in integer) return integer;
    pragma INTERFACE(C, MSGRCV);
    pragma INTERFACE NAME(MSGRCV, "msgrcv");
    function MSGCTL(MSGID : in integer;
                    CMD : in integer;
                    BUFF : in SYSTEM.address) return integer;
    pragma INTERFACE(C, MSGCTL);
    pragma INTERFACE_NAME(MSGCTL, "msgctl");
end RODB_Test_Data;
```

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b					
		d	b					
		d	b					
r rrr	oooo	ddd d	b bbb		t	eeee	ssss	ttttt
rr r	o o	d dd	bb b		t	e e	s s	t
r	o o	d d	b b		t	eeeeee	ss	t
r	o o	d d	b b		t	e	ss	t
r	o o	d dd	bb b		t t	e e	s s	t t
r	oooo	ddd d	b bbb		tt	eeee	ssss	tt

---

Job: rodb\_test6.ada  
Date: Sat Mar 28 21:48:39 1992

```

-- This is the performace test for IPC message queue
with TEXT_IO, CALENDAR, SYSTEM, RODB_Test_Data, POSIX_UNSAFE_PROCESS_PRIMITIVE
use TEXT_IO, CALENDAR, SYSTEM, RODB_Test_Data, POSIX_UNSAFE_PROCESS_PRIMITIVE
with POSIX_PROCESS_IDENTIFICATION, POSIX_PROCESS_PRIMITIVES;
use POSIX_PROCESS_IDENTIFICATION, POSIX_PROCESS_PRIMITIVES;
procedure RODB_Test6 is

    -- Constant definition
    MSGKEY1      : constant integer := 99;
    MSGKEY2      : constant integer := 100;
    MTEXT_SIZE   : constant integer := 500;
    MSG_LEN      : constant integer := 10;

    -- Data type definition
    type MSGForm_Type is record
        MType : integer;
        MText : string(1..MTEXT_SIZE);
    end record;

    -- Package instantiation
    package INT_IO is new TEXT_IO.INTEGER_IO(integer);
    package FIX_IO is new TEXT_IO.FIXED_IO(duration);

    -- Variable definition
    My_PID          : POSIX_PROCESS_IDENTIFICATION.process_id;
    My_Status       : POSIX_PROCESS_PRIMITIVES.termination_status;
    Msgid1          : integer;
    Msgid2          : integer;
    Flag            : integer;
    Number_Of_Times : integer;
    Start_Time      : CALENDAR.time;
    Finish_Time     : CALENDAR.time;
    Outfile         : file_type;
    Sender_Msg      : MSGForm_Type;
    Receiver_Msg    : MSGForm_Type;
    My_Msg          : string(1..MSG_LEN);
    Your_Msg        : string(1..MSG_LEN);
    My_Response     : string(1..MSG_LEN);
    Your_Response   : string(1..MSG_LEN);

    -- Exception definition
    Msg_Exception : exception;
begin

    -- Input the number of times from user
    put("Enter the number of times: ");
    INT_IO.get(Number_Of_Times);

    -- Create two IPC Message Queues(request and response queues)
    Msgid1 := RODB_Test_Data.MSGGET(MSGKEY1, 1023);
    if Msgid1 = -1 then
        put_line("Error in MSGGET.");
        raise Msg_Exception;
    end if;
    Msgid2 := RODB_Test_Data.MSGGET(MSGKEY2, 1023);
    if Msgid2 = -1 then
        put_line("Error in MSGGET.");
        raise Msg_Exception;
    end if;

```

```

-- Create two processes (Sender and Receiver)
My_PID := POSIX_UNSAFE_PROCESS_PRIMITIVES.fork; -- fork a receiver child
if My_PID = POSIX_PROCESS_IDENTIFICATION.NULL_PROCESS_ID then -- Child
    Start_Time := CALENDAR.clock;
    for I in 1..Number_Of_Times loop
        Flag:=RODB_Test_Data.MSGRCV(Msgid1, Receiver_Msg'address, MSG_LEN, 1, 0);
        if Flag = -1 then
            put_line("Error in MSGRCV.");
            raise Msg_Exception;
        end if;
        My_Msg(1..MSG_LEN) := Receiver_Msg.MText(1..MSG_LEN);
        My_Response(1..MSG_LEN) := "Hello guys";
        Sender_Msg.MType := 1;
        Sender_Msg.MText(1..MSG_LEN) := My_Response(1..MSG_LEN);
        Flag := RODB_Test_Data.MSGSND(Msgid2, Sender_Msg'address, MSG_LEN, 0);
        if Flag = -1 then
            put_line("Error in MSGSND.");
            raise Msg_Exception;
        end if;
    end loop;
    Finish_Time := CALENDAR.clock;

    Delay 20.0;    -- Wait for parent to manipulate the message queue

    -- Output the result to a file
    create(Outfile, out_file, "rodbcompl.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "Number_Of_Iterations    Times");
    INT_IO.put(Outfile, Number_Of_Times);
    FIX_IO.put(Outfile, Finish_Time-Start_Time);
    new_line(Outfile);
    close(Outfile);

    POSIX_PROCESS_PRIMITIVES.exit_process; -- Child Exits
else
    -- Parent
    Start_Time := CALENDAR.clock;
    for I in 1..Number_Of_Times loop
        Your_Msg(1..MSG_LEN) := "Hi world!";
        Sender_Msg.MType := 1;
        Sender_Msg.MText(1..MSG_LEN) := Your_Msg(1..MSG_LEN);
        Flag := RODB_Test_Data.MSGSND(Msgid1, Sender_Msg'address, MSG_LEN, 0);
        if Flag = -1 then
            put_line("Error in MSGSND.");
            raise Msg_Exception;
        end if;
        Flag:=RODB_Test_Data.MSGRCV(Msgid2, Receiver_Msg'address, MSG_LEN, 1,0);
        if Flag = -1 then
            put_line("Error in MSGRCV.");
            raise Msg_Exception;
        end if;
        Your_Response(1..MSG_LEN) := Receiver_Msg.MText(1..MSG_LEN);
    end loop;
    Finish_Time := CALENDAR.clock;
end if;
POSIX_PROCESS_PRIMITIVES.wait_for_child(My_Status);

-- Output the result to a file
create(Outfile, out_file, "rodbcomp2.out",
    form=>"world=>read, owner=>read_write");

```

```

put_line(Outfile, "Number_of Iterations    Times");
INT_IO.put(Outfile, Number_of_Times);
FIX_IO.put(Outfile, Finish_Time-Start_Time);
new_line(Outfile);
close(Outfile);

-- Remove the IPC Message Queues(request and response queues)
Flag := RODB_Test_Data.MSGCTL(Msgid1, 0, SYSTEM.NULL_ADDRESS);
if Flag = -1 then
    put_line("Error in MSGCTL.");
    raise Msg_Exception;
end if;
Flag := RODB_Test_Data.MSGCTL(Msgid2, 0, SYSTEM.NULL_ADDRESS);
if Flag = -1 then
    put_line("Error in MSGCTL.");
    raise Msg_Exception;
end if;

exception
    when Msg_Exception =>
        put_line("Program terminates abnormally.");
    when others =>
        put_line("Other exception in main program.");
end RODB_Test6;

```

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b						1
		d	b						11
		d	b						1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d dd	bb b	c c	o o	m m m	pp p		1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp		11111
							p		
							p		
							p		

Job: rodbcompl.out  
Date: Sat Mar 28 21:31:31 1992

Number_Of_Iterations	Times
1000	1.29565
2000	2.51501
3000	3.75342
4000	5.08582
5000	6.33374



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b					22222
		d	b					2 2
		d	b					2
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp	2
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p	222
r	o o	d d	b b	c	o o	m m m	p p	2
r	o o	d d	b b	c	o o	m m m	p p	2
r	o o	d dd	bb b	c c	o o	m m m	pp p	2
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp	2222222
							p	
							p	
							p	

Job: rodbcomp2.out  
Date: Sat Mar 28 21:31:44 1992

Number_Of_Iterations	Times
1000	1.29565
Number_Of_Iterations	Times
2000	2.52454
Number_Of_Iterations	Times
3000	3.75342
Number_Of_Iterations	Times
4000	5.08582
Number_Of_Iterations	Times
5000	6.33374



---

Copies of this publication have been deposited with the Texas State Library in compliance with the State Depository Law.

---

# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

**Appendices C1 - C5**

*Prepared for*

***NASA/JSC Data Management Systems  
for Space Station Freedom (SSF)***

Prepared by Co-Principal Investigators:

**T. F. Leibfried Jr.  
Sadegh Davari**  
*University of Houston-Clear Lake*

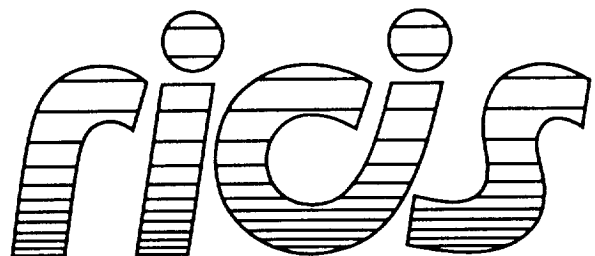
**Swami Natarajan  
Wei Zhao**  
*Texas A&M University*

Research Associates:

**Libin Wu**  
*University of Houston-Clear Lake*

**Gary Smith**  
*Texas A& M University*

April 1992



*Research Institute for Computing and Information Systems  
University of Houston-Clear Lake*

---

**F • I • N • A • L     R • E • P • O • R • T**

## ***The RICIS Concept***

---

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

**APPENDICES  
TO  
FINAL REPORT**

**VOL 2 OF 3  
APPENDICES C (1-5)**





## **Appendix C-1**

**Uncontested RODB Reads**

**and**

**Uncontested RODB Writes**

**All Protection Mechanisms are Enabled as Semaphore Protection and Disabling of Preemption to Guarantee Atomic Semaphore Transitions in Place.**



```

      t
      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o      o      o      t
r      o      o      o      t
r      o      o      o      t
r      o      o      o      t  t
r      oooo      oooo      tt

```

```

      d
      d
      d
r rrr      eeee      aaaa      ddd d
rr   r      e   e      a   d   dd
r      eeeee      aaaaa      d   d
r      e   e      a   a      d   d
r      e   e      a   aa      d   dd
r      eeee      aaaa a      ddd d

```

```

m m mm      eeee
mm m m      e   e
m m m      eeeee
m m m      e
.. m m m      e   e
.. m m m      eeee

```

Job: read.me  
Date: Mon Mar 30 00:56:21 1992

This directory stores all the files to build up a RODB "attribute" components. The protection mechanism is that locking is set at the RODB level. During the lock setting, there is a prevention of preemption used to protect the semaphore test-and-lock operation to insure atomicity. This is done inside a C function by using the fast\_setprio system call. There is only one set of three UNIX semaphores in the whole system but a total of seven array operations on these three semaphores. Before actual READING, a set of five semaphore operations are imposed upon the three semaphores, two of which are repeated. The reason for the five (with two repeats) is to simulate what might have to be done if this were to be implemented with the current Lynx OS (where the kernel is preemptable) and no prevention of preemption were to be done; ( This is done in a later test). After actually reading the RODB one semaphore operation is imposed on the semaphores. Before actual WRITING there are two levels of operations: write desire and write lock. For write "desire" only one semaphore operation is imposed on the semaphores and for write "lock" an array of four semaphore operations are imposed on the semaphores. After actual writing, a set of two semaphore operations are imposed on the semaphores. A TEST IS DONE TO MEASURE HOW LONG IT TAKES FOR 1000 TO 1000 READS AND WRITES. THE RESULTS ARE IN FILE rodbcomp1.dat for reads. THE RESULTS ARE IN FILE rodbcomp2.dat for writes. This test does not involve contention since the reads and writes are done in separate runs.

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d	b			88888	1
			d	b	t	t	8 8	11
			d	b	t	t	8 8	1 1
r rrr	oooo	ddd d	b bbb	ttttt	ssss	ttttt	8 8	1
rr r	o o	d dd	bb b	t	s s	t	88888	1
r	o o	d d	b b	t	ss	t	8 8	1
r	o o	d d	b b	t	ss	t	8 8	1
r	o o	d dd	bb b	t t	s s	t t	8 8	1
r	oooo	ddd d	b bbb	tt	ssss	tt	88888	11111

Job: rodbtst81.ada  
Date: Mon Mar 30 00:48:25 1992

```

-- This is the reading test program, (uncontested reads)
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure Rodbtst81 is

    -- Constant definitions
    ATTR_SIZE    : constant integer := 200;
    RESULT_SIZE  : constant integer := 10;

    -- Data type definition
    type Result_Type is record
        Loops : integer;
        Times : duration;
    end record;

    -- Package instantiation
    package INT_IO is new TEXT_IO.INTEGER_IO(integer);
    package FIX_IO is new TEXT_IO.FIXED_IO(duration);
    package RCDT renames RODB_Component_Data_Types;
    package ROBCP renames RODB_Component;

    -- Variable definitions
    Length          : integer;
    Number_Of_Times : integer;
    Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
    Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
    Start_Time      : CALENDAR.time;
    Finish_Time     : CALENDAR.time;
    Results         : array(1..RESULT_SIZE) of Result_Type;
    Outfile         : file_type;

begin
    ROBCP.Load_Comps("rodbcomp.dat"); -- load the test RODB
    Length := 1;
    Addr_List(1) := 0;
    Number_Of_Times := 1000; -- inner loop iterations initialization
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock; -- get the start time for inner loop
        for J in 1..Number_Of_Times loop
            ROBCP.Read_Attrs(Addr_List, Length, Attr_List); -- Read RODB
        end loop;
        Finish_Time := CALENDAR.clock; -- record the end time
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time); -- Store data
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;

    -- Output the result to a file now that test is over
    create(Outfile, out_file, "rodbcompl.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, " rodbcompl.dat ");
    put_line(Outfile, "Test NO NO_Of_Iterations Times");
    for I in 1..RESULT_SIZE loop
        INT_IO.put(Outfile, I, width => 5);
        INT_IO.put(Outfile, Results(I).Loops);
        FIX_IO.put(Outfile, Results(I).Times);
        new_line(Outfile);
    end loop;
    close(Outfile);
exception
    when others =>

```

```
put_line("Main program exception");  
end Rodbtst81;
```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

		d	b						1
		d	b						11
		d	b						1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d dd	bb b	c c	o o	m m m	pp p		1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp		11111
							p		
							p		
							p		

Job: rodbcomp1.out  
Date: Mon Mar 30 00:57:09 1992



Test NO	NO_Of_Iterations	Times
1	1000	1.43866
2	2000	2.90466
3	3000	4.33374
4	4000	5.80927
5	5000	7.24799
6	6000	8.71387
7	7000	10.14307
8	8000	11.62805
9	9000	13.04773
10	10000	14.53265

```

ll      b
l      i b      i
l      b
l      ii b bbb ii n nnn
l      i bb b i nn n
l      i b b i n n
l      i b b i n n
l      i bb b i n n
lll     iii b bbb iii n n

```

```

              d b
              d b
              d b
r rrr         oooo ddd d b bbb t t 88888 22222
rr r         o   o d dd bb b tttt ssss ttttt 8 8 2 2
r           o   o d d b b t t s s t t 8 8 2 2
r           o   o d d b b t t ss ss t t 88888 222
r           o   o d dd bb b t t s s t t 8 8 2
r           oooo ddd d b bbb tt ssss tt 88888 2222222

```

Job: rodibtst82.ada  
 Date: Mon Mar 30 00:48:26 1992

```
-- This is the writing test program, uncontested writes
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure Rodbtst82 is
```

```
-- Constant definitions
```

```
ATTR_SIZE    : constant integer := 200;
RESULT_SIZE  : constant integer := 10;
```

```
-- Data type definition
```

```
type Result_Type is record
    Loops : integer;
    Times : duration;
end record;
```

```
-- Package instantiation
```

```
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package ROBCP renames RODB_Component;
```

```
-- Variable definition
```

```
Length       : integer;
Number_Of_Times : integer;
Addr_List    : RCDT.Pos_List_Type(1..ATTR_SIZE);
Attr_List    : RCDT.Attr_List_Type(1..ATTR_SIZE);
Start_Time   : CALENDAR.time;
Finish_Time  : CALENDAR.time;
Results      : array(1..RESULT_SIZE) of Result_Type;
Outfile      : file_type;
```

```
begin
```

```
    ROBCP.Load_Comps("rodbcomp.dat"); -- load the RODB Components
```

```
    Length := 1;
```

```
    Addr_List(1) := 0;
```

```
    Attr_List(1) := (Type_ID => 0, Int_Value => 200);
```

```
    Number_Of_Times := 1000;
```

```
    for I in 1..RESULT_SIZE loop
```

```
        Start_Time := CALENDAR.clock; -- log the beginning
```

```
        for J in 1..Number_Of_Times loop
```

```
            ROBCP.Write_Attrs(Addr_List, Length, Attr_List); -- write
        end loop;
```

```
        Finish_Time := CALENDAR.clock; -- log the end
```

```
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
```

```
        Number_Of_Times := Number_Of_Times + 1000;
```

```
    end loop;
```

```
-- Output the result to a file now that test is over
```

```
create(Outfile, out_file, "rodbcomp2.out",
      form=>"world=>read, owner=>read_write");
```

```
put_line(Outfile, " rodbcomp2.out ");
```

```
put_line(Outfile, "Test NO  NO_Of_Iterations  Times");
```

```
for I in 1..RESULT_SIZE loop
```

```
    INT_IO.put(Outfile, I, width => 5);
```

```
    INT_IO.put(Outfile, Results(I).Loops);
```

```
    FIX_IO.put(Outfile, Results(I).Times);
```

```
    new_line(Outfile);
```

```
end loop;
```

```
close(Outfile);
```

```
exception
```

```
when others =>  
    put_line("Main program exception");  
end Rodbtst82;
```

ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d d	b b	c	o o	m m m	pp p
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

  -- Package renaming
  package RCDT renames Rodb_Component_Data_Types;

  -- Exception definition
  Shm_Exception : exception;
  Shm_Outrange  : exception;
  Sem_Exception : exception;

  -- Read attributes from RODB components
  procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                       Length     : in      integer;
                       Attr_List  : in out RCDT.Attr_List_Type);

  -- Write attributes to RODB components
  procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length     : in      integer;
                        Attr_List  : in      RCDT.Attr_List_Type);

  -- Print out the semaphore values
  procedure Print_Sems;

  -- Load RODB components from a disk file
  procedure Load_Comps(Filename : in      string);

  -- Save RODB components to a disk file
  procedure Save_Comps(Filename : in      string);

  -- Shutdown the RODB components
  procedure Shutdown_Comps;

end RODB_COMPONENT;

```

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

			d	b					22222
			d	b					2
			d	b					2
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		2
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p	222	
r	o o	d d	b b	c	o o	m m m	p p	2	
r	o o	d d	b b	c	o o	m m m	p p	2	
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp	222222	
							p		
							p		
							p		

Job: rodbcomp2.out  
Date: Mon Mar 30 00:48:24 1992

rodbcomp2.out

Test NO	NO_Of_Iterations	Times
1	1000	1.73431
2	2000	3.57086
3	3000	5.30518
4	4000	7.09534
5	5000	8.88556
6	6000	10.66620
7	7000	12.40057
8	8000	14.22754
9	9000	15.92505
10	10000	17.76160



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c	o o	m m m	p p
r	o o	d d	b b	c	o o	m m m	pp p
r	o o	d dd	bb b	c c	o o	m m m	pp ppp
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

---

Job: rodb\_component\_data\_types\_.ada  
Date: Mon Mar 30 00:51:26 1992

-- This package provides the constants, instantiated packages, system calls  
-- and C functions interfaces to C language for RODB COMPONENT package.

with TEXT\_IO, SYSTEM;

use TEXT\_IO, SYSTEM;

package RODB\_COMPONENT\_DATA\_TYPES is

-- Constants

INT\_SIZE : constant integer := 10;

CHAR\_SIZE : constant integer := 10;

BOOL\_SIZE : constant integer := 10;

FLT\_SIZE : constant integer := 10;

SHMKEY : constant integer := 99;

SEMKEY : constant integer := 100;

SHM\_SIZE : constant integer := INT\_SIZE\*4+CHAR\_SIZE+BOOL\_SIZE+4\*FLT\_SIZE;

CHAR\_OFFSET : constant integer := INT\_SIZE\*4;

BOOL\_OFFSET : constant integer := CHAR\_OFFSET + CHAR\_SIZE\*1;

FLT\_OFFSET : constant integer := BOOL\_OFFSET + BOOL\_SIZE\*1;

-- Data types

type Attr\_Type (Type\_ID : integer := 0) is record

case Type\_ID is

when 0 =>

Int\_Value : integer;

when 1 =>

Char\_Value : character;

when 2 =>

Bool\_Value : boolean;

when 3 =>

Flt\_Value : float;

when others =>

null;

end case;

end record;

type Attr\_List\_Type is array(integer range <>) of Attr\_Type;

type Pos\_List\_Type is array(integer range <>) of integer;

-- Package instantiation

package INT\_IO is new TEXT\_IO.INTEGER\_IO(integer);

package BOOL\_IO is new TEXT\_IO.ENUMERATION\_IO(boolean);

package FLT\_IO is new TEXT\_IO.FLOAT\_IO(float);

function FINT is new system.fetch\_from\_address(integer);

function FCHAR is new system.fetch\_from\_address(character);

function FBOOL is new system.fetch\_from\_address(boolean);

function FFLT is new system.fetch\_from\_address(float);

procedure AINT is new system.assign\_to\_address(integer);

procedure ACHAR is new system.assign\_to\_address(character);

procedure ABOOL is new system.assign\_to\_address(boolean);

procedure AFLT is new system.assign\_to\_address(float);

-- Shared memory system call interface

function SHMGET(KEY : in integer;

SIZE : in integer;

FLAG : in integer) return integer;

pragma INTERFACE(C, SHMGET);

pragma INTERFACE NAME(SHMGET, "shmget");

function SHMAT(SHMID : in integer;

SHMADDR : in system.address;

FLAG : in integer) return system.address;

pragma INTERFACE(C, SHMAT);

pragma INTERFACE\_NAME(SHMAT, "shmat");

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD    : in integer;
                BUFF   : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

-- Semaphore system call and C function interface
function SEMGET(KEY    : in integer;
                NSEMS  : in integer;
                FLAG    : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

end RODB_Component_Data_Types;

```

```

                d  b
                d  b
                d  b
r rrr      oooo      ddd d  b bbb
rr  r      o    o    d  dd  bb  b
r          o    o    d    d  b    b
r          o    o    d    d  b    b
r          o    o    d  dd  bb  b
r          oooo      ddd d  b bbb

```

Job: rodb\_component .ada  
Date: Mon Mar 30 00:54:23 1992

```
with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is
```

```
-- Package renaming
package RCDT renames Rodb_Component_Data_Types;
```

```
-- Exception definition
Shm_Exception : exception;
Shm_Outrange  : exception;
Sem_Exception  : exception;
```

```
-- Read attributes from RODB components
procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                     Length    : in      integer;
                     Attr_List : in out RCDT.Attr_List_Type);
```

```
-- Write attributes to RODB components
procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                     Length    : in      integer;
                     Attr_List : in      RCDT.Attr_List_Type);
```

```
-- Print out the semaphore values
procedure Print_Sems;
```

```
-- Load RODB components from a disk file
procedure Load_Comps(Filename : in      string);
```

```
-- Save RODB components to a disk file
procedure Save_Comps(Filename : in      string);
```

```
-- Shutdown the RODB components
procedure Shutdown_Comps;
```

```
end RODB_COMPONENT;
```

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	bb b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c c	o o	m m m	p p
r	o o	d d	b b	c c	o o	m m m	p p
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

---

Job: rodb\_component.ada  
Date: Mon Mar 30 00:48:30 1992

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components simulating array of handles read

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is

```

```

    Temp : system.address;

```

```

    Flag : integer;

```

```

begin

```

```

    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION; -- more efficient in C program

```

```

    Flag := RCDT.READBEG(Semid);

```

```

    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION; -- in C program

```

```

    if Flag = -1 then

```

```

        raise Sem_Exception;

```

```

    end if;

```

```

    for I in 1..Length loop

```

```

        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;

```

```

        end if;

```

```

        Temp := Shmaddr + system.offset(Addr_List(I));

```

```

        if (Addr_List(I) < RCDT.CHAR_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));

```

```

        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));

```

```

        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));

```

```

        else

```

```

            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));

```

```

        end if;

```

```

    end loop;

```

```

    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION; -- again done in C program

```

```

    Flag := RCDT.READEND(Semid); -- This is a C function

```

```

    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;

```

```

    if Flag = -1 then

```

```

        raise Sem_Exception;

```

```

    end if;

```

```

end Read_Attrs;

```

```

-- Write attributes to RODB components again simulating array of handles

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

    Temp : system.address;

```

```

    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION; /* In C program for efficiency */
    Flag := RCDT.WRITEBEG(Semid); -- also a C function
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION; /* Also in C */
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1.. Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            RCDT.AINT(Temp, Attr_List(I).Int_Value);
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
        else
            RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
        end if;
    end loop;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION; /* In C program */
    Flag := RCDT.WRITEEND(Semid); -- A C function
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION; /* In C program */
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
    Flag : integer;
begin
    Flag := RCDT.SEMPRINT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--     Number_Of_Integers
--     Position1 Integer1
--     Position2 Integer2
--     ...
--     Number_Of_Characters
--     Position1 Character1
--     Position2 Character2
--     ...
--     Number_Of_Booleans
--     Position1 Boolean1
--     Position2 Boolean2
--     ...
--     Number_Of_Floats
--     Position1 Float1
--     Position2 Float2
--     ...
procedure Load_Comps(Filename : in string) is

```



```

Infile : FILE_TYPE;
Temp   : system.address;
Flag   : integer;
begin

    open(Infile, in_file, Filename);

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        RCDT.ACHAR(Temp, 'X');
    end loop;
    Load_Chars(Infile);

    -- Initialize RODB Boolean Component
    for I in 1..RCDT.BOOL_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        RCDT.ABOOL(Temp, true);
    end loop;
    Load_Bools(Infile);

    -- Initialize RODB Float Component
    for I in 1..RCDT.FLT_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        RCDT.AFLT(Temp, 0.0);
    end loop;
    Load_Flts(Infile);

    close(Infile);

    Flag := RCDT.SEMSINIT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;

exception
    when name_error =>
        put_line("File cannot be opened.");
        put_line("Loading components fails!");
    when data_error | end_error =>
        put_line("File format is incompatible.");
        put_line("Loading components fails!");
    when Sem_Exception =>
        put_line("Semaphore cannot be initialized.");
        raise Sem_Exception;
    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file
-- The structure of the disk file is as following:
--     Number_Of_Integers

```

```

--      Position1  Integer1
--      Position2  Integer2
--      ...
--      Number_Of_Characters
--      Position1  Character1
--      Position2  Character2
--      ...
--      Number_Of_Booleans
--      Position1  Boolean1
--      Position2  Boolean2
--      ...
--      Number_Of_Floats
--      Position1  Float1
--      Position2  Float2
--      ...
procedure Save_Comps(Filename : in      string) is
    Outfile : FILE_TYPE;
begin
    if Filename /= "" then
        create(Outfile, out_file, Filename,
               form=>"world=>read, owner=>read_write");

        Save_Ints(Outfile);
        Save_Chars(Outfile);
        Save_Bools(Outfile);
        Save_Flts(Outfile);
        close(Outfile);
    else
        Save_Ints(TEXT_IO.standard_output);
        Save_Chars(TEXT_IO.standard_output);
        Save_Bools(TEXT_IO.standard_output);
        Save_Flts(TEXT_IO.standard_output);
    end if;
exception
    when constraint_error =>
        put_line("RODB Components data collapsed.");
        put_line("Saving components fails!");
    when others =>
        put_line("Unknown exception.");
        put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
    Flag : integer;
begin
    Flag := RCDT.SHMDT(Shmaddr);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SEMSRMV(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Shutdown_Comps;

pragma page;

```

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Int    : integer;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
    RCDT.AINT(Temp_Addr, Temp_Int);
  end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Char   : character;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    get(Infile, Temp_Char);           -- Skip a space
    get(Infile, Temp_Char);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
    RCDT.ACHAR(Temp_Addr, Temp_Char);
  end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Bool   : boolean;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    BOOL_IO.get(Infile, Temp_Bool);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
      raise Shm_Outrange;
    end if;
  end loop;
end Load_Bools;
```

```

    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

```

```

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is

```

```

    Length : Integer;
    Temp_Pos : integer;
    Temp_Flt : float;
    Temp_Addr : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

```

```

pragma page;

```

```

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in FILE_TYPE) is

```

```

    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

```

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in FILE_TYPE) is

```

```

    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Chars;

```

end Save\_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file  
procedure Save\_Bools(Outfile : in FILE\_TYPE) is

Temp\_Addr : system.address;  
begin  
put(Outfile, "Number Of Booleans is: ");  
Int\_IO.put(Outfile, RCDT.BOOL\_SIZE);  
new\_line(Outfile);  
for I in 1..RCDT.BOOL\_SIZE loop  
put(Outfile, "Boolean number ");  
Int\_IO.put(Outfile, I-1, width => 5);  
put(Outfile, ":");  
Temp\_Addr := Shmaddr + system.offset(RCDT.BOOL\_OFFSET + I-1);  
Bool\_IO.put(Outfile, RCDT.FBOOL(Temp\_Addr));  
new\_line(Outfile);  
end loop;  
end Save\_Bools;

-- Save all the floats from RODB Float Component to a disk file  
procedure Save\_Flts(Outfile : in FILE\_TYPE) is

Temp\_Addr : system.address;  
begin  
put(Outfile, "Number Of Floats is ");  
Int\_IO.put(Outfile, RCDT.FLT\_SIZE);  
new\_line(Outfile);  
for I in 1..RCDT.FLT\_SIZE loop  
put(Outfile, "Float number ");  
Int\_IO.put(Outfile, I-1);  
put(Outfile, ":");  
Temp\_Addr := Shmaddr + system.offset(RCDT.FLT\_OFFSET + (I-1)\*4);  
Flt\_IO.put(Outfile, RCDT.FFLT(Temp\_Addr));  
new\_line(Outfile);  
end loop;  
end Save\_Flts;

pragma page;

begin

Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM\_SIZE, 1023);  
if Shmid = -1 then  
raise Shm\_Exception;  
end if;  
Shmaddr := RCDT.SHMAT(Shmid, system.null\_address, 0);  
-- if Shmaddr = system.null\_address then  
-- raise Shm\_Exception;  
-- end if;  
Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);  
if Semid = -1 then  
raise Sem\_Exception;  
end if;  
end Rodb\_Component;

```

r rrr      eeee      aaaa      ddd d      b      bbb      eeee      ggg g      cccc
rr  r      e   e      a      d   dd      bb   b      e   e      g   gg      c   c
r          eeeeee      aaaaa      d   d      b   b      eeeee      g   g      c   c
r          e   e      a      a      d   d      b   b      e   e      g   gg      c   c
r          e   e      a      aa     d   dd     bb   b      e   e      g   gg      c   c
r          eeee      aaaa a      ddd d      b bbb      eeee      gggg g      cccc
                                     g   g
                                     gggg

```

```
Job:  readbeg.c
Date:  Mon Mar 30 01:00:08 1992
```

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t    my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform three semaphore operations, two of them twice (see read.me) */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writers */
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writers */
    one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writers */
    one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writers */
    one_of_n_readers[4] = RREAD_START; /* Prevent writers in */
    flag = semop(semid, one_of_n_readers, 5); /* Lock the critical section */
    if (flag == -1) {
        perror("readbeg fails: ");
    }

    /* Lower the priority to normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

      d
      d
      d
      d
r rrr  eeee  aaaa  ddd d  eeee  n nnn  ddd d  cccc
rr   r  e   e  a   a  d   dd  e   e  nn   n  d   dd  c   c
r      eeeee  aaaaa  d   d  eeeee  n   n  d   d  c   c
r      e   e  a   a  d   d  e   e  n   n  d   d  c   c
r      eeee  aaaa a  ddd d  eeee  n   n  ddd d  ..  cccc
      ..

```

Job: readend.c  
Date: Mon Mar 30 01:00:18 1992



```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);          /* Unlock critical section */
    if (flag == -1) {
        perror("readend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

      i      t      b
      t      b
      t      b
w      w  r rrr  ii  ttttt  eeee  b bbb  eeee  ggg g
w  w  w  rr  r  i  t  e   e  bb  b  e   e  g  gg
w  w  w  r      i  t  eeeee  b   b  eeeee  g  g
w  w  w  r      i  t  e      b   b  e      g  g
w  w  w  r      i  t  e   e  bb  b  e   e  g  gg
  ww ww  r      iii  tt  eeee  b bbb  eeee  ggg g
                                     g
                                     g
                                     gggg
..
..

```

Job: writebeg.c  
Date: Mon Mar 30 01:00:32 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START     = { 0, 1, 0};
struct sembuf WWRITE_LOCK     = { 1, 1, 0};
struct sembuf WWRITE_DESIRE   = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding readers */
    sole_writer[2] = WREAD_START;      /* preventing succeeding writers */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r        oooo      oooo      tt

```

```

                                     d
                                     d
                                     d
      i      t      ttttt      eeee      eeee      n nnn      ddd d
w      w      r rrr      ii      t      e   e      e   e      nn   n      d   dd
w w w      rr   r      i      t      eeeee      eeeee      n   n      d   d
w w w      r        i      t      e       e       n   n      d   d
w w w      r        i      t t      e   e      e   e      n   n      d   dd
ww ww      r      iii      tt      eeee      eeee      n   n      ddd d

```

..  
..

Job: writeend.c  
Date: Mon Mar 30 01:00:58 1992

```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

i          i          t
ii         n nnn      ii         ttttt
i          nn         i          t
i          n          i          t
i          n          i          t
i          n          i          t  t
iii        n          iii        tt

```

```
Job:  semsinit.c
Date:  Mon Mar 30 01:01:24 1992
```

```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

ssss  eeee  m m mm  ssss  r rrr  m m mm  v      v      cccc
s    s  e    e  mm m  m  s    s  rr  r  mm m  m  v      v      c  c
ss      eeeee  m  m  m  ss      r      m  m  m  v      v      c
      ss      e    e  m  m  m  ss      r      m  m  m  v  v      c
s    s  e    e  m  m  m  s    s  r      m  m  m  v  v      c
ssss  eeee  m  m  m  ssss  r      m  m  m  v      v      c

```

Job: semsrmv.c  
Date: Mon Mar 30 01:01:37 1992



```
/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          oooo      oooo      tt

```

```

                                     i
                                     t
                                     t
ssss      eeee      m m mm      p ppp      r rrr      ii      n nnn      ttttt
s  s      e  e      mm m m      pp  p      rr  r      i      nn  n      t
ss        eeeeeee  m  m  m      p    p      r          i      n    n      t
  ss      e        m  m  m      p    p      r          i      n    n      t
s  s      e  e      m  m  m      pp  p      r          i      n    n      t  t
ssss      eeee      m  m  m      p ppp      r          iii     n    n      tt      ..
                                     p
                                     p
                                     p

```

Job: semprint.c  
Date: Mon Mar 30 01:01:48 1992

```
/*File:semprint.c This is semaphore print subroutine to print semaphore values*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semprint(semid)
    int semid;
{
    short outarray[3];
    int    flag;
    void perror();
    int    i;

    flag = semctl(semid, 3, GETALL, outarray);
    if (flag == -1) {
        perror("semprint fails: ");
    }
    for (i=0; i<3; ++i) {
        printf("Semaphore %d has the value of %d\n", i, outarray[i]);
    }
    return(flag);
}
```



## **Appendix C-2**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**Semaphore Protection in Place but no Disabling of Preemption (i.e. no raising/lowering of priorities)**



r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

			d
			d
			d
r rrr	eeee	aaaa	ddd d
rr r	e e	a	d dd
r	eeeeee	aaaaa	d d
r	e	a a	d d
r	e e	a aa	d dd
r	eeee	aaaa a	ddd d

	m m mm	eeee
	mm m m	e e
	m m m	eeeeee
	m m m	e
..	m m m	e e
..	m m m	eeee

Job: read.me  
Date: Wed Apr 1 19:12:17 1992

This directory stores all the files to build up RODB "attribute" components. The protection mechanism is that locking is set at the RODB level. During the lock setting, there is NO PREVENTION OF PREEMPTION. That is to say tests are performed with semaphores which are not guaranteed to have atomic execution. Actually in the case at hand that is not a problem because the reads and writes are done in separate runs so that they are uncontested. Before the actual reads are begun the semaphores are tested and set. First, a set of five semaphore operations are imposed upon the semaphores. After the actual reading of the RODB component one semaphore operation is imposed on the semaphores. In the case of writing, before actually writing to the RODB component, there are two levels of operations to be imposed on the semaphores. These operations are to express a desire to write or write desire and a locking of a semaphore or write lock. To achieve the setting of write desire, only one semaphore operation will be imposed on the semaphores and for After actually writing, a set of two more semaphore operations are imposed upon the semaphores.

A TEST IS DONE TO MEASURE HOW LONG IT TAKES FOR 1000 TO 10000 READS and then the same for WRITES.

THE RESULT IS IN FILE rodbcomp91.dat for reading. THE RESULT FOR WRITING IS IN FILE rodbcomp92.dat



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          o  o      o  o      tt
r          oooo      oooo

```

```

                        d  b
                        d  b
                        d  b
r rrr      oooo      ddd d  b bbb      ttttt      ssss      ttttt      99999      1
rr  r      o  o      d  dd  bb  b      t      s  s      t      9  9      11
r          o  o      d  d  b  b      t      s  s      t      9  9      1 1
r          o  o      d  d  b  b      t      ss      t      9  9      1
r          o  o      d  dd  bb  b      t  t      s  s      t  t      9  9      1
r          oooo      ddd d  b bbb      tt      ssss      tt      9999      11111

```

Job: rodhtst91.ada  
Date: Wed Apr 1 19:30:59 1992

```
-- This is the reading test program
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure Rodbtst91 is
```

```
-- Constant definitions
ATTR_SIZE    : constant integer := 200;
RESULT_SIZE  : constant integer := 10;
```

```
-- Data type definition
type Result_Type is record
    Loops : integer;
    Times : duration;
end record;
```

```
-- Package instantiation
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package RODBCP renames RODB_Component;
```

```
-- Variable definition
Length          : integer;
Number_Of_Times : integer;
Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
Start_Time      : CALENDAR.time;
Finish_Time     : CALENDAR.time;
Results         : array(1..RESULT_SIZE) of Result_Type;
Outfile         : file_type;
```

```
begin
    RODBCP.Load_Comps("rodbcomp.dat");
    Length := 1;
    Addr_List(1) := 0;
    Number_Of_Times := 1000;
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock;
        for J in 1..Number_Of_Times loop
            RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
        end loop;
        Finish_Time := CALENDAR.clock;
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;
```

```
-- Output the result to a file
create(Outfile, out_file, "rodbcomp91.out",
    form=>"world=>read, owner=>read_write");
put_line(Outfile, "NO      Number_Of_Iterations      Times");
for I in 1..RESULT_SIZE loop
    INT_IO.put(Outfile, I, width => 5);
    INT_IO.put(Outfile, Results(I).Loops);
    FIX_IO.put(Outfile, Results(I).Times);
    new_line(Outfile);
end loop;
close(Outfile);
exception
    when others =>
        put_line("Main program exception");
```

end Rodbtst91;

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb      ttttt      ssss      ttttt      99999      22222
rr   r    o   o    d   dd   bb   b      t      s   s      t      9   9      2
r      o   o    d   d   b     b      t      s       t      9   9      2
r      o   o    d   d   b     b      t      ss      t      9   9      2
r      o   o    d   dd   bb   b      t t     s   s      t t     9   9      2
r      oooo      ddd d   b bbb      tt      ssss      tt      9999      2222222

```

Job: rodbtst92.ada  
 Date: Wed Apr 1 19:33:27 1992

```
-- This is the reading test program
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure Rodbtst92 is
```

```
-- Constant definitions
ATTR_SIZE    : constant integer := 200;
RESULT_SIZE  : constant integer := 10;
```

```
-- Data type definition
type Result_Type is record
    Loops : integer;
    Times : duration;
end record;
```

```
-- Package instantiation
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package RODBCP renames RODB_Component;
```

```
-- Variable definition
Length          : integer;
Number_Of_Times : integer;
Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
Start_Time      : CALENDAR.time;
Finish_Time     : CALENDAR.time;
Results         : array(1..RESULT_SIZE) of Result_Type;
Outfile         : file_type;
```

```
begin
    RODBCP.Load_Comps("rodbcomp.dat");
    Length := 1;
    Addr_List(1) := 0;
    Attr_List(1) := (Type_ID => 0, Int_Value => 200);
    Number_Of_Times := 1000;
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock;
        for J in 1..Number_Of_Times loop
            RODBCP.Write_Attrs(Addr_List, Length, Attr_List);
        end loop;
        Finish_Time := CALENDAR.clock;
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;
```

```
-- Output the result to a file
create(Outfile, out_file, "rodbcomp92.out",
      form=>"world=>read, owner=>read_write");
```

```
put_line(Outfile, "NO      Number_Of_Iterations      Times");
for I in 1..RESULT_SIZE loop
    INT_IO.put(Outfile, I, width => 5);
    INT_IO.put(Outfile, Results(I).Loops);
    FIX_IO.put(Outfile, Results(I).Times);
    new_line(Outfile);
end loop;
close(Outfile);
```

```
exception
when others =>
```

```
    put_line("Main program exception");  
end Rodbtst92;
```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb  cccc  oooo  m m mm  p ppp
rr  r  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p
r      o  o  d  d  b  b  c  c  o  o  m m m  p  p
r      o  o  d  dd  bb  b  c  c  o  o  m m m  pp  p
r      oooo  ddd d  b bbb  cccc  oooo  m m m  p ppp  ..
      p
      p
      p

```

Job: rodbcomp.dat  
Date: Wed Apr 1 19:12:40 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0



```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

                        d   b
                        d   b
                        d   b
r rrr      oooo      ddd d   b bbb      cccc      oooo      m m mm      p ppp      99999
rr   r    o   o    d   dd   bb   b   c   c      o   o    mm m m      pp   p      9   9
r      o   o    d   d   b   b   c   c      o   o    m m m m      p   p      999999
r      o   o    d   d   b   b   c   c      o   o    m m m m      p   p      9
r      o   o    d   dd   bb   b   c   c      o   o    m m m m      pp   p      9
r      oooo      ddd d   b bbb      cccc      oooo      m m m      p ppp      9999
                                           p
                                           p
                                           p

```

Job: rodbcomp91.out  
Date: Wed Apr 1 19:31:54 1992

NO	Number_Of_Iterations	Times
1	1000	0.54224
2	2000	1.03815
3	3000	1.53406
4	4000	2.11304
5	5000	2.57220
6	6000	3.10492
7	7000	3.64716
8	8000	4.16211
9	9000	4.68530
10	10000	5.17163

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t t
r          oooo      oooo      tt

```

```

                        d  b
                        d  b
                        d  b
r rrr      oooo      ddd d  b bbb      cccc      oooo      m m mm      p ppp      99999
rr  r      o  o      d  dd  bb  b      c  c      o  o      mm m m      pp  p      9 9
r          o  o      d  d  b  b      c          o  o      m m m      p  p      999999
r          o  o      d  d  b  b      c          o  o      m m m      p  p      9
r          o  o      d  dd  bb  b      c  c      o  o      m m m      pp  p      9
r          oooo      ddd d  b bbb      cccc      oooo      m m m      p ppp      9999
                                           p
                                           p
                                           p

```

Job: rodbcomp92.out  
Date: Wed Apr 1 19:33:10 1992

NO	Number_Of_Iterations	Times
1	1000	0.73297
2	2000	1.47552
3	3000	2.16211
4	4000	2.88556
5	5000	3.58173
6	6000	4.37061
7	7000	5.04767
8	8000	5.76294
9	9000	6.52313
10	10000	7.20984

r rrr	oooo	oooo	t t tttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
r	oooo	ddd d	b bbb

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp p
c	o o	m m m	p p
c	o o	m m m	p p
c c	o o	m m m	pp p
cccc	oooo	m m m	p ppp
			p
			p
			p

Job: rodb\_component\_data\_types\_.ada  
Date: Wed Apr 1 19:13:20 1992

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY   : in integer;
                  SIZE  : in integer;
                  FLAG   : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID  : in integer;
                 SHMADDR : in system.address;
                 FLAG    : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD    : in integer;
                BUFF   : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

-- Semaphore system call and C function interface

```

function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

end RODB\_Component\_Data\_Types;

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr  r      o  o      o  o
r          o  o      o  o
r          o  o      o  o
r          o  o      o  o
r          oooo      oooo

```

```

      d  b
      d  b
      d  b
      d  b bbb
      d dd bb  b
      d d  b   b
      d dd bb  b
      ddd d b bbb

r rrr      oooo      ddd d
rr  r      o  o      d  dd
r          o  o      d  d
r          o  o      d  d
r          o  o      d  dd
r          oooo      ddd d

```

```

      cccc      oooo      m m mm      p ppp
      c  c      o  o      mm m  m      pp  p
      c  c      o  o      m  m  m      p  p
      c  c      o  o      m  m  m      p  p
      c  c      o  o      m  m  m      pp  p
      cccc      oooo      m  m  m      p ppp
                                     p
                                     p
                                     p

```

---

Job: rodb\_component\_.ada  
Date: Wed Apr 1 19:13:42 1992



```
with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is
```

```
-- Package renaming
package RCDT renames Rodb_Component_Data_Types;
```

```
-- Exception definition
Shm_Exception : exception;
Shm_Outrange  : exception;
Sem_Exception  : exception;
```

```
-- Read attributes from RODB components
procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                    Length      : in      integer;
                    Attr_List   : in out RCDT.Attr_List_Type);
```

```
-- Write attributes to RODB components
procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                    Length      : in      integer;
                    Attr_List   : in      RCDT.Attr_List_Type);
```

```
-- Print out the semaphore values
procedure Print_Sems;
```

```
-- Load RODB components from a disk file
procedure Load_Comps(Filename : in      string);
```

```
-- Save RODB components to a disk file
procedure Save_Comps(Filename : in      string);
```

```
-- Shutdown the RODB components
procedure Shutdown_Comps;
```

```
end RODB_COMPONENT;
```

```

                d  b
                d  b
                d  b
r rrr      oooo      ddd d  b bbb
rr  r      o    o    d  dd  bb  b
r          o    o    d    d  b    b
r          o    o    d    d  b    b
r          o    o    d  dd  bb  b
r          oooo      ddd d  b bbb

```

Job: rodb\_component.ada  
Date: Wed Apr 1 19:14:09 1992

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables
Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms
procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components
procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READBEG(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READEND(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components
procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

Temp : system.address;
Flag : integer;
begin
-- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
Flag := RCDT.WRITEBEG(Semid);
-- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
if Flag = -1 then
raise Sem_Exception;
end if;
for I in 1.. Length loop
if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
raise Shm_Outrange;
end if;
Temp := Shmaddr + system.offset(Addr_List(I));
if (Addr_List(I) < RCDT.CHAR_OFFSET) then
RCDT.AINT(Temp, Attr_List(I).Int_Value);
elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
else
RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
end if;
end loop;
-- delay 10.0;
-- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
Flag := RCDT.WRITEEND(Semid);
-- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
if Flag = -1 then
raise Sem_Exception;
end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
Flag : integer;
begin
Flag := RCDT.SEMPRINT(Semid);
if Flag = -1 then
raise Sem_Exception;
end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--      Number_Of_Integers
--      Position1 Integer1
--      Position2 Integer2
--      ...
--      Number_Of_Characters
--      Position1 Character1
--      Position2 Character2
--      ...
--      Number_Of_Booleans
--      Position1 Boolean1
--      Position2 Boolean2
--      ...
--      Number_Of_Floats
--      Position1 Float1
--      Position2 Float2

```

```

--
...
procedure Load Comps(Filename : in    string) is
    Infile : FILE_TYPE;
    Temp    : system.address;
    Flag    : integer;
begin
    open(Infile, in_file, Filename);

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        RCDT.ACHAR(Temp, 'X');
    end loop;
    Load_Chars(Infile);

    -- Initialize RODB Boolean Component
    for I in 1..RCDT.BOOL_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        RCDT.ABOOL(Temp, true);
    end loop;
    Load_Bools(Infile);

    -- Initialize RODB Float Component
    for I in 1..RCDT.FLT_SIZE loop
        Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        RCDT.AFLT(Temp, 0.0);
    end loop;
    Load_Flts(Infile);

    close(Infile);

    Flag := RCDT.SEMSINIT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;

exception
    when name_error =>
        put_line("File cannot be opened.");
        put_line("Loading components fails!");
    when data_error | end_error =>
        put_line("File format is incompatible.");
        put_line("Loading components fails!");
    when Sem_Exception =>
        put_line("Semaphore cannot be initialized.");
        raise Sem_Exception;
    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
          form=>"world=>read, owner=>read_write");
    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint_error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```

pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Int    : integer;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
    RCDT.AINT(Temp_Addr, Temp_Int);
  end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Char   : character;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    get(Infile, Temp_Char);           -- Skip a space
    get(Infile, Temp_Char);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
    RCDT.ACHAR(Temp_Addr, Temp_Char);
  end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Bool   : boolean;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    BOOL_IO.get(Infile, Temp_Bool);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

```

-- Load all the floats from a disk file to RODB Float Component

```

procedure Load_Flts(Infile : in    FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

```

pragma page;

-- Save all the integers from RODB Integer Component to a disk file

procedure Save\_Ints(Outfile : in FILE\_TYPE) is

```

    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

-- Save all the characters from RODB Character Component to a disk file

procedure Save\_Chars(Outfile : in FILE\_TYPE) is

```

    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```



```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      eeee      aaaa      ddd d      b bbb      eeee      ggg g      cccc
rr   r    e   e      a      d   dd      bb   b      e   e      g   gg      c   c
r      eeeee      aaaaa      d   d      b   b      eeeee      g   g      c   c
r      e   e      a   a      d   d      b   b      e   e      g   gg      c   c
r      e   e      a   aa      d   dd      bb   b      e   e      ggg g      c   c
r      eeee      aaaa a      ddd d      b bbb      eeee      gggg      cccc
                                     g
                                     g
                                     gggg

```

Job: readbeg.c  
Date: Wed Apr 1 19:35:08 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK  = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)      /* Now Modified to remove priority raising */
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption NOW DISABLED ! */
    /* my tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */

    /* Perform three semaphore operations where two are repeated */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK;      /* Wait for no more writer */
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE;    /* Wait for no more writer */
    one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK;      /* Wait for no more writer */
    one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE;    /* Wait for no more writer */
    one_of_n_readers[4] = RREAD_START;              /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 5);      /* Lock the critical section */
    if (flag == -1) {
        perror("readbeg fails: ");
    }

    /* Lower the priority to the normal ALSO DISABLED ! */
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

                                d
                                d
                                d
                                d
r rrr  eeee  aaaa  ddd d  eeee  n nnn  ddd d  cccc
rr   r  e   e  a   a  d  dd  e   e  nn   n  d  dd  c   c
r      eeeee  aaaaa  d   d  eeeee  n   n  d   d  c   c
r      e   e  a   aa  d   dd  e   e  n   n  d   dd  c   c
r      eeee  aaaa a  ddd d  eeee  n   n  ddd d  ..  cccc
                                ..

```

Job: readend.c  
Date: Wed Apr 1 19:35:17 1992

```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption NOW DISABLED ! */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);      /* Unlock critical section */
    if (flag == -1) {
        perror("readend fails: ");
    }

    /* Lower the priority to the normal Not needed so DISABLED ! */
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          oooo      oooo      tt

```

```

      i      t      b
      t      b
      t      b
w      w      r rrr      ii      ttttt      eeee      b bbb      eeee      ggg g
w w w rr  r      i      t      e e      bb  b      e e      g gg
w w w r      i      t      eeeee      b  b      eeeee      g g
w w w r      i      t      e      b  b      e      g g
w w w r      i      t t      e e      bb  b      e e      g gg
ww ww r      iii      tt      eeee      b bbb      eeee      ggg g
                                     g
                                     g
                                     gggg

```

..  
..

Job: writebeg.c  
Date: Wed Apr 1 19:35:27 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS    = { 0, 0, 0};
struct sembuf WREAD_START        = { 0, 1, 0};
struct sembuf WWRITE_LOCK        = { 1, 1, 0};
struct sembuf WWRITE_DESIRE      = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption, NOW DISABLED ! */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding readers */
    sole_writer[2] = WREAD_START;      /* preventing succeeding writers */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }

    /* Lower the priority to the normal NOW not needed so DISABLED ! */
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r        oooo      oooo      tt

```

```

                                     d
                                     d
                                     d
w      w  r rrr      ii      t      ttttt      eeee      eeee      n nnn      ddd d
w  w  w  rr   r      i      t      e   e      e   e      nn   n      d   dd
w  w  w  r        i      t      eeeee      eeeee      n   n      d   d
w  w  w  r        i      t      e       e      n   n      d   d
w  w  w  r        i      t t   t      e   e      e   e      n   n      d   dd
  ww ww  r      iii      tt      eeee      eeee      n   n      ddd d      ..

```

Job: writeend.c  
 Date: Wed Apr 1 19:35:39 1992



```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption NOW DISABLED ! */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }

    /* Lower the priority to the normal, NOW DISABLED ! */
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

				i		i	t	
							t	
ssss	eeee	m m mm	ssss	ii	n nnn	ii	ttttt	
s s	e e	mm m m	s s	i	nn n	i	t	
ss	eeeeee	m m m	ss	i	n n	i	t	
ss	e e	m m m	ss	i	n n	i	t	
s s	e e	m m m	s s	i	n n	i	t t	..
ssss	eeee	m m m	ssss	iii	n n	iii	tt	..

Job: semsinit.c  
Date: Wed Apr 1 19:36:11 1992

```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r        oooo     oooo     tt

```

```

ssss      eeee      m m mm      ssss      r rrr      m m mm      v      v      cccc
s   s    e   e    mm m m    s   s    rr   r    mm m m    v      v      c   c
ss      eeeeeee  m m m    ss      r        m m m    v      v      c   c
   ss    e       m m m      ss      r        m m m    v      v      c   c
s   s    e   e    m m m    s   s    r        m m m    v v      ..   c   c
ssss      eeee    m m m    ssss      r        m m m      v      ..   cccc

```

Job: semsrmv.c  
Date: Wed Apr 1 19:35:52 1992

```
/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

                                     i
                                     t
                                     t
ssss  eeee  m m mm  p ppp  r rrr  ii  n nnn  ttttt
s    s  e    e  mm m m  pp  p  rr  r  i  nn  n  t
ss      eeeee  m m m m  p    p  r      i  n  n  t
ss      e      m m m m  p    p  r      i  n  n  t
s    s  e    e  m m m m  pp  p  r      i  n  n  t  t
ssss  eeee  m m m  p ppp  r      iii  n  n  tt  ..
                                     p
                                     p
                                     p

```

Job: semprint.c  
Date: Wed Apr 1 19:37:08 1992

```

/*File:semprint.c This is semaphore print subroutine to print semaphore values*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semprint(semid)
    int semid;
{
    short outarray[3];
    int    flag;
    void    perror();
    int    i;

    flag = semctl(semid, 3, GETALL, outarray);
    if (flag == -1) {
        perror("semprint fails: ");
    }
    for (i=0; i<3; ++i) {
        printf("Semaphore %d has the value of %d\n", i, outarray[i]);
    }
    return(flag);
}

```





## **Appendix C-3**

**Uncontested RODB Reads**

**and**

**Uncontested RODB Writes**

**Two Disabling and Re-enabling Pairs for each Read or Write. No Protection for Actual Reads or Writes.**



```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      oooo  oooo  tt

```

```

                        d  b
                        d  b
                        d  b
r rrr  oooo  ddd d  b bbb  ttttt  ssss  ttttt  AAA  1
rr  r  o  o  d  dd  bb  b  t      s  s      A  A  11
r      o  o  d  d  b  b  b  t      ss      A  A  1 1
r      o  o  d  d  b  b  b  t      ss      A  A  1
r      o  o  d  dd  bb  b  t  t  s  s      A  A  1
r      oooo  ddd d  b bbb  tt  ssss  tt  A  A  1111

```

Job: rodbtstA1.ada  
Date: Thu Apr 9 13:17:43 1992

```

-- This is the reading test program without semaphores

with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure RodbtstA1 is

    -- Constant definitions
    ATTR_SIZE    : constant integer := 200;
    RESULT_SIZE  : constant integer := 10;

    -- Data type definition
    type Result_Type is record
        Loops : integer;
        Times : duration;
    end record;

    -- Package instantiation
    package INT_IO is new TEXT_IO.INTEGER_IO(integer);
    package FIX_IO is new TEXT_IO.FIXED_IO(duration);
    package RCDT renames RODB_Component_Data_Types;
    package RODBCP renames RODB_Component;

    -- Variable definition
    Length          : integer;
    Number_Of_Times : integer;
    Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
    Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
    Start_Time      : CALENDAR.time;
    Finish_Time     : CALENDAR.time;
    Results         : array(1..RESULT_SIZE) of Result_Type;
    Outfile         : file_type;

begin
    RODBCP.Load_Comps("rodbcomp.dat");
    Length := 1;
    Addr_List(1) := 0;
    Number_Of_Times := 1000;
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock;
        for J in 1..Number_Of_Times loop -- Two priority raise/lower per read
            RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
        end loop;
        Finish_Time := CALENDAR.clock;
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;

    -- Output the result to a file
    create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "NO      Number_Of_Iterations      Times");
    for I in 1..RESULT_SIZE loop
        INT_IO.put(Outfile, I, width => 5);
        INT_IO.put(Outfile, Results(I).Loops);
        FIX_IO.put(Outfile, Results(I).Times);
        new_line(Outfile);
    end loop;
    close(Outfile);
exception
    when others =>

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t  t
r      oooo  oooo  tt

```

```

      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr   r  e   e    a  d  dd
r      eeeee  aaaaa  d  d
r      e    a    a  d  d
r      e   e  a   aa  d  dd
r      eeee  aaaa a  ddd d

```

```

m m mm  eeee
mm m m  e   e
m m m  eeeee
m m m  e
.. m m m  e   e
.. m m m  eeee

```

Job: read.me  
Date: Thu Apr 9 13:24:03 1992

This directory stores all the files to build up RODB "attribute" components. The protection mechanism is that locking is set at the RODB level. To achieve lock setting, a prevention-of-preemption system call is used. This is done inside a C function which has been called from Ada. The mechanism used is the fast `setprio` system call which is supposed to change priority expeditiously. WE ARE TRYING TO SEE HOW MUCH OVERHEAD THERE IS FOR THE PRIORITY CHANGE SYSTEM CALL. A TEST IS DONE TO MEASURE HOW LONG IT TAKES FOR 1000 TO 10000 READS AND WRITES (two priority raise/lower per Read or Write event). THE RESULT IS IN FILE `rodbcomp1.out` for reading. THE RESULT IS IN FILE `rodbcomp2.out` for writes.

```
put_line("Main program exception");  
end RodbtstA1;
```

```

      1
     11
    1 1
      1
      1
      1
      1
      1
    11111

```

[illegible]

Job: rodbcomp1.out  
Date: Thu Apr 9 13:21:57 1992



NO	Number_Of_Iterations	Times
1	1000	1.00000
2	2000	2.00000
3	3000	3.00000
4	4000	4.00952
5	5000	5.00000
6	6000	6.00952
7	7000	7.01910
8	8000	8.00952
9	9000	9.00000
10	10000	10.00952

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          o      o      o      o      t
r          oooo      oooo      tt

```

```

                        d      b
                        d      b
                        d      b
r rrr      oooo      ddd d      b bbb      t      t      AAA      22222
rr  r      o      o      d      dd      bb  b      ttttt      ssss      ttttt      A      A      2      2
r          o      o      d      d      b      b      t      s      s      t      A      A      2
r          o      o      d      d      b      b      t      ss      t      A      A      2
r          o      o      d      dd      bb  b      t  t      s      s      t  t      A      A      2
r          oooo      ddd d      b bbb      tt      ssss      tt      A      A      2222222

```

Job: rodbtstA2.ada  
 Date: Thu Apr 9 13:17:55 1992

```

-- This is the writing test program without semaphores
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure RodbtstA2 is

  -- Constant definitions
  ATTR_SIZE    : constant integer := 200;
  RESULT_SIZE  : constant integer := 10;

  -- Data type definition
  type Result_Type is record
    Loops : integer;
    Times : duration;
  end record;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package FIX_IO is new TEXT_IO.FIXED_IO(duration);
  package RCDT renames RODB_Component_Data_Types;
  package ROBCP renames RODB_Component;

  -- Variable definition
  Length          : integer;
  Number_Of_Times : integer;
  Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
  Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
  Start_Time      : CALENDAR.time;
  Finish_Time     : CALENDAR.time;
  Results         : array(1..RESULT_SIZE) of Result_Type;
  Outfile         : file_type;

begin
  ROBCP.Load_Comps("rodbcomp.dat");
  Length := 1;
  Addr_List(1) := 0;
  Attr_List(1) := (Type_ID => 0, Int_Value => 200);
  Number_Of_Times := 1000;
  for I in 1..RESULT_SIZE loop
    Start_Time := CALENDAR.clock;
    for J in 1..Number_Of_Times loop -- Two priority raise/lower per write
      ROBCP.Write_Attrs(Addr_List, Length, Attr_List);
    end loop;
    Finish_Time := CALENDAR.clock;
    Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
    Number_Of_Times := Number_Of_Times + 1000;
  end loop;

  -- Output the result to a file
  create(Outfile, out_file, "rodbcomp2.out",
        form=>"world=>read, owner=>read_write");
  put_line(Outfile, "NO      Number_Of_Iterations      Times");
  for I in 1..RESULT_SIZE loop
    INT_IO.put(Outfile, I, width => 5);
    INT_IO.put(Outfile, Results(I).Loops);
    FIX_IO.put(Outfile, Results(I).Times);
    new_line(Outfile);
  end loop;
  close(Outfile);
exception
  when others =>

```

```
put_line("Main program exception");  
end RodbtstA2;
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

                        d   b
                        d   b
                        d   b
r rrr      oooo      ddd d   b bbb      cccc      oooo      m m mm      p ppp
rr   r    o   o    d   dd   bb   b   c   c      o   o    mm m m      pp   p
r      o   o    d   d   b       b   c       o   o    m m m      p   p
r      o   o    d   d   b       b   c       o   o    m m m      p   p
r      o   o    d   dd   bb   b   c       o   o    m m m      pp   p
r      oooo      ddd d   b bbb      cccc      oooo      m m m      p ppp
                                           p
                                           p
                                           p

```

22222  
2 2  
2  
2  
222  
2  
2  
2  
222222

Job: rodbcomp2.out  
Date: Thu Apr 9 13:22:06 1992

NO	Number_Of_Iterations	Times
1	1000	1.05725
2	2000	2.10492
3	3000	3.15259
4	4000	4.21930
5	5000	5.32294
6	6000	6.31470
7	7000	7.37195
8	8000	8.46594
9	9000	9.47681
10	10000	10.57086

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          oooo      oooo      tt

```

```

      d      b
      d      b
      d      b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr  r      o  o      d dd      bb b      c  c      o  o      mm m m      pp p
r          o  o      d d      b b      c      o  o      m m m      p p
r          o  o      d d      b b      c      o  o      m m m      p p
r          o  o      d dd      bb b      c  c      o  o      m m m      pp p
r          oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp
                                           p
                                           p
                                           p

```

Job: rodbcomp.dat  
Date: Thu Apr 9 13:22:20 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0



```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r        oooo     oooo     tt

```

```

      d   b
      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb
rr   r    o   o    d   dd  bb   b
r        o   o    d   d   b     b
r        o   o    d   d   b     b
r        o   o    d   dd  bb   b
r        oooo     ddd d   b bbb

```

```

      cccc      oooo      m m mm      p ppp
c   c    o   o    mm m m      pp   p
c   c    o   o    m m m      p   p
c   c    o   o    m m m      p   p
c   c    o   o    m m m      pp   p
cccc      oooo      m m m      p ppp
                                     p
                                     p
                                     p

```

---

Job: rodb\_component\_data\_types\_.ada  
 Date: Thu Apr 9 13:18:38 1992

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY   : in integer;
                  SIZE  : in integer;
                  FLAG  : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID  : in integer;
                 SHMADDR : in system.address;
                 FLAG    : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

-- Semaphore system call and C function interface
function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

end RODB_Component_Data_Types;

```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
	oooo	oooo	tt

			d	b
			d	b
			d	b
r rrr	oooo	ddd d	b bbb	
rr r	o o	d dd	bb b	
r	o o	d d	b b	
r	o o	d d	b b	
r	o o	d dd	bb b	
	oooo	ddd d	b bbb	

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp p
c c	o o	m m m	p p
c c	o o	m m m	pp p
cccc	oooo	m m m	p ppp
			p
			p
			p

Job: rodb\_component\_ada  
Date: Thu Apr 9 13:18:56 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception  : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in      RCDT.Attr_List_Type);

    -- Print out the semaphore values
    procedure Print_Sems;

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end RODB_COMPONENT;

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb
rr  r  o  o  d  dd  bb  b
r      o  o  d  d  b  b
r      o  o  d  d  bb  b
r      oooo  ddd d  b bbb

```

```

      cccc  oooo  m m mm  p ppp
c  c  o  o  mm m m  pp  p
c  c  o  o  m m m  p  p
c  c  o  o  m m m  pp  p
c  c  o  o  m m m  p ppp
      cccc  oooo  m m m  p
                        p
                        p

```

---

Job: rodb\_component.ada  
 Date: Thu Apr 9 13:19:13 1992

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is

```

```

    Temp : system.address;

```

```

    Flag : integer;

```

```

begin

```

```

    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;

```

```

    Flag := RCDT.READBEG(Semid);

```

```

    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;

```

```

    if Flag = -1 then

```

```

        raise Sem_Exception;

```

```

    end if;

```

```

    for I in 1..Length loop

```

```

        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then

```

```

            raise Shm_Outrange;

```

```

        end if;

```

```

        Temp := Shmaddr + system.offset(Addr_List(I));

```

```

        if (Addr_List(I) < RCDT.CHAR_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));

```

```

        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));

```

```

        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));

```

```

        else

```

```

            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));

```

```

        end if;

```

```

    end loop;

```

```

    -- delay 10.0;

```

```

    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;

```

```

    Flag := RCDT.READEND(Semid);

```

```

    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;

```

```

    if Flag = -1 then

```

```

        raise Sem_Exception;

```

```

    end if;

```

```

end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

Temp : system.address;
Flag : integer;
begin
-- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
Flag := RCDT.WRITEBEG(Semid);
-- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
if Flag = -1 then
raise Sem_Exception;
end if;
for I in 1.. Length loop
if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
raise Shm_Outrange;
end if;
Temp := Shmaddr + system.offset(Addr_List(I));
if (Addr_List(I) < RCDT.CHAR_OFFSET) then
RCDT.AINT(Temp, Attr_List(I).Int_Value);
elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
else
RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
end if;
end loop;
-- delay 10.0;
-- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
Flag := RCDT.WRITEEND(Semid);
-- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
if Flag = -1 then
raise Sem_Exception;
end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
Flag : integer;
begin
Flag := RCDT.SEMPRINT(Semid);
if Flag = -1 then
raise Sem_Exception;
end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--      Number_Of_Integers
--      Position1 Integer1
--      Position2 Integer2
--      ...
--      Number_Of_Characters
--      Position1 Character1
--      Position2 Character2
--      ...
--      Number_Of_Booleans
--      Position1 Boolean1
--      Position2 Boolean2
--      ...
--      Number_Of_Floats
--      Position1 Float1
--      Position2 Float2

```



```
/*File:semprint.c This is semaphore print subroutine to print semaphore values*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semprint(semid)
    int semid;
{
    short outarray[3];
    int    flag;
    void    perror();
    int    i;

    flag = semctl(semid, 3, GETALL, outarray);
    if (flag == -1) {
        perror("semprint fails: ");
    }
    for (i=0; i<3; ++i) {
        printf("Semaphore %d has the value of %d\n", i, outarray[i]);
    }
    return(flag);
}
```

```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */
    /* Don't bother to raise a priority that wasn't lowered by writebeg */

    /* Perform the semaphore operation */ /* Commented out here */
    /* sole_writer[0] = WREAD_END; */ /* Allow writer in */
    /* sole_writer[1] = WWRITE_UNLOCK; */ /* Allow reader in */
    /* flag = semop(semid, sole_writer, 2); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("writeend fails: ");
    } */

    /* Lower the priority to the normal */ /* Undo what writebeg.c did */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      oooo  oooo  tt

```

```

                                     d
                                     d
                                     d
                                     d
w      w  r rrr  ii  ttttt  eeee  eeee  n nnn  ddd d
w w w  rr  r  i  t  e  e  e  e  nn  n  d  dd
w w w  r  i  t  e  e  e  e  n  n  d  d
w w w  r  i  t  e  e  e  e  n  n  d  dd
w w w  r  i  t  t  e  e  e  e  n  n  d  dd  ..
  ww ww  r  iii  tt  eeee  eeee  n  n  ddd d  ..

```

Job: writeend.c  
Date: Thu Apr 9 13:32:24 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0};
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE    = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Make write request by doing a semaphore operation */ /* Commented out */
    /* flag = semop(semid, &WWRITE_DESIRE, 1); */
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    } /*

    /* Perform four semaphore operations */ /* All semops commented out */
    /* sole_writer[0] = WWAIT_NO_READERS; */ /* Wait for no more readers */
    /* sole_writer[1] = WWRITE_LOCK; */ /* preventing succeeding readers */
    /* sole_writer[2] = WREAD_START; */ /* preventing succeeding writers */
    /* sole_writer[3] = WIN_PROGRESS_WRITE; */ /* Cancel the write-request */
    /* flag = semop(semid, sole_writer, 4); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    } */

    /* Lower the priority to the normal */ /* Commented out here */
    /* fast_setprio(my_tid, my_prio); */ /* postpone lowering priority til end of write */

    return 0;
}

```

```

      t
      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        oooo      oooo      tt

```

```

      i      t      b
      t      b
      t      b
w      w  r rrr      ii      ttttt      eeee      b bbb      eeee      ggg g
w  w  w  rr   r      i      t      e   e      bb   b      e   e      g   gg
w  w  w  r        i      t      eeeee     b   b      eeeee     g   g
w  w  w  r        i      t      e         b   b      e         g   g
w  w  w  r        i      t  t      e   e      bb   b      e   e      g   gg
  ww ww  r      iii      tt      eeee      b bbb      eeee      ggg g      ..
                                     g         gggg      g         gggg      ..

```

Job: writebeg.c  
Date: Thu Apr 9 13:33:07 1992

```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t  my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */ /* Commented out already done by readbeg.c */

    /* Perform the semaphore operation */ /* Semaphore ops commented out here */
    /* flag = semop(semid, &RREAD_END, 1); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("readend fails: ");
    } */

    /* Lower the priority to the normal */ /* Undo what readbeg.c did */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      oooo  oooo  tt

```

```

                        d
                        d
                        d
r rrr  eeee  aaaa  ddd d  eeee  n nnn  ddd d  cccc
rr   r  e   e  a   a  d  dd  e   e  nn   n  d  dd  c   c
r      eeeee  aaaaa  d   d  eeeee  n   n  d   d  c
r      e   e  a   a  d   d  e   e  n   n  d   d  c
r      e   e  a   aa  d  dd  e   e  n   n  d  dd  ..  c
r      eeee  aaaa a  ddd d  eeee  n   n  ddd d  ..  cccc

```

Job: readend.c  
Date: Thu Apr 9 13:30:47 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform three semaphore operations */ /* All semaphores commented out */
    /* one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[4] = RREAD_START; */ /* Prevent writer in */
    /* flag = semop(semid, one_of_n_readers, 5); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("readbeg fails: ");
    } */

    /* Lower the priority to the normal */ /* not done here for test rodbtstB1 */
    /* fast_setprio(my_tid, my_prio); */

    return 0; /* Commented out the lowering of the priority, that done by readend */
}

```



```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o    o  o    o  t
r      o    o  o    o  t
r      o    o  o    o  t
r      o    o  o    o  t t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  eeee  aaaa  ddd d  b bbb  eeee  ggg g  cccc
rr  r  e    e  a    d  dd  bb  b  e    e  g  gg  c    c
r      eeeee  aaaaa  d    d  b    b  eeeee  g  g  c
r      e    e  a    a  d    d  b    b  e    e  g  gg  c
r      e    e  a    aa  d  dd  bb  b  e    e  ggg g  ..  c
r      eeee  aaaa a  ddd d  b bbb  eeee  ggg g  ..  cccc
                                g  g
                                gggg

```

Job: readbeg.c  
Date: Sun Apr 5 01:18:06 1992

```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

```

```

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

```

```
pragma page;
```

```

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

```

-- Load all the floats from a disk file to RODB Float Component

```

procedure Load_Flts(Infile : in      FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

```

pragma page;

-- Save all the integers from RODB Integer Component to a disk file

```

procedure Save_Ints(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

-- Save all the characters from RODB Character Component to a disk file

```

procedure Save_Chars(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```

pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Int    : integer;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
    RCDT.AINT(Temp_Addr, Temp_Int);
  end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Char   : character;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    get(Infile, Temp_Char);           -- Skip a space
    get(Infile, Temp_Char);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
    RCDT.ACHAR(Temp_Addr, Temp_Char);
  end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Bool   : boolean;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    BOOL_IO.get(Infile, Temp_Bool);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
```

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
           form=>"world=>read, owner=>read_write");

    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint_error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```

```

--      ...
procedure Load_Comps(Filename : in      string) is
  Infile : FILE_TYPE;
  Temp   : system.address;
  Flag   : integer;
begin

  open(Infile, in_file, Filename);

  -- Initialize RODB Integer Component
  for I in 1..RCDT.INT_SIZE loop
    Temp := Shmaddr + system.offset((I-1)*4);
    RCDT.AINT(Temp, 0);
  end loop;
  Load_Ints(Infile);

  -- Initialize RODB Character Component
  for I in 1..RCDT.CHAR_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
  end loop;
  Load_Chars(Infile);

  -- Initialize RODB Boolean Component
  for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
  end loop;
  Load_Bools(Infile);

  -- Initialize RODB Float Component
  for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
  end loop;
  Load_Flts(Infile);

  close(Infile);

  Flag := RCDT.SEMSINIT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;

exception
  when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
  when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
  when Sem_Exception =>
    put_line("Semaphore cannot be initialized.");
    raise Sem_Exception;
  when others =>
    put_line("Unknown exception.");
    put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```

```

Temp : system.address;
Flag : integer;
begin
  -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
  Flag := RCDT.WRITEBEG(Semid);
  -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
  if Flag = -1 then
    raise Sem_Exception;
  end if;
  for I in 1.. Length loop
    if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp := Shmaddr + system.offset(Addr_List(I));
    if (Addr_List(I) < RCDT.CHAR_OFFSET) then
      RCDT.AINT(Temp, Attr_List(I).Int_Value);
    elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
      RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
    elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
      RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
    else
      RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
    end if;
  end loop;
  -- delay 10.0;
  -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
  Flag := RCDT.WRITEEND(Semid);
  -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
  Flag : integer;
begin
  Flag := RCDT.SEMPRINT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--       Number_Of_Integers
--       Position1 Integer1
--       Position2 Integer2
--       ...
--       Number_Of_Characters
--       Position1 Character1
--       Position2 Character2
--       ...
--       Number_Of_Booleans
--       Position1 Boolean1
--       Position2 Boolean2
--       ...
--       Number_Of_Floats
--       Position1 Float1
--       Position2 Float2

```

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READBEG(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READEND(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```



```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t  t
r        oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb
rr   r      o   o      d   dd  bb   b
r        o   o      d   d   b     b
r        o   o      d   d   b     b
r        o   o      d   dd  bb   b
r        oooo      ddd d   b bbb

```

```

      cccc      oooo      m m mm      p ppp
c   c      o   o      mm m   m      pp   p
c        o   o      m   m   m      p    p
c        o   o      m   m   m      p    p
c   c      o   o      m   m   m      pp   p
cccc      oooo      m   m   m      p ppp
                                     p
                                     p
                                     p

```

---

Job: rodb\_component.ada  
 Date: Sun Apr 5 01:17:38 1992

```
with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is
```

```
-- Package renaming
```

```
package RCDT renames Rodb_Component_Data_Types;
```

```
-- Exception definition
```

```
Shm_Exception : exception;
```

```
Shm_Outrange  : exception;
```

```
Sem_Exception : exception;
```

```
-- Read attributes from RODB components
```

```
procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                     Length     : in      integer;
                     Attr_List  : in out RCDT.Attr_List_Type);
```

```
-- Write attributes to RODB components
```

```
procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                     Length     : in      integer;
                     Attr_List  : in      RCDT.Attr_List_Type);
```

```
-- Print out the semaphore values
```

```
procedure Print_Sems;
```

```
-- Load RODB components from a disk file
```

```
procedure Load_Comps(Filename : in      string);
```

```
-- Save RODB components to a disk file
```

```
procedure Save_Comps(Filename : in      string);
```

```
-- Shutdown the RODB components
```

```
procedure Shutdown_Comps;
```

```
end RODB_COMPONENT;
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t t
r        oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d      b bbb
rr   r      o   o      d dd      bb b
r        o   o      d d        b b
r        o   o      d d        b b
r        o   o      d dd      bb b
r        oooo      ddd d      b bbb

```

```

      cccc      oooo      m m mm      p ppp
c   c      o   o      mm m m      pp p
c   c      o   o      m m m      p p
c   c      o   o      m m m      p p
c   c      o   o      m m m      pp p
      cccc      oooo      m m m      p ppp
                                   p
                                   p
                                   p

```

---

Job: rodb\_component\_ada  
 Date: Sun Apr 5 01:16:27 1992

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

-- Semaphore system call and C function interface

```

function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

end RODB\_Component\_Data\_Types;

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

-- Constants
INT_SIZE      : constant integer := 10;
CHAR_SIZE     : constant integer := 10;
BOOL_SIZE     : constant integer := 10;
FLT_SIZE      : constant integer := 10;
SHMKEY        : constant integer := 99;
SEMKEY        : constant integer := 100;
SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
CHAR_OFFSET   : constant integer := INT_SIZE*4;
BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

-- Data types
type Attr_Type (Type_ID : integer := 0) is record
  case Type_ID is
    when 0 =>
      Int_Value : integer;
    when 1 =>
      Char_Value : character;
    when 2 =>
      Bool_Value : boolean;
    when 3 =>
      Flt_Value  : float;
    when others =>
      null;
  end case;
end record;
type Attr_List_Type is array(integer range <>) of Attr_Type;
type Pos_List_Type is array(integer range <>) of integer;

-- Package instantiation
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
package FLT_IO is new TEXT_IO.FLOAT_IO(float);
function FINT is new system.fetch_from_address(integer);
function FCHAR is new system.fetch_from_address(character);
function FBOOL is new system.fetch_from_address(boolean);
function FFLT is new system.fetch_from_address(float);
procedure AINT is new system.assign_to_address(integer);
procedure ACHAR is new system.assign_to_address(character);
procedure ABOOL is new system.assign_to_address(boolean);
procedure AFLT is new system.assign_to_address(float);

-- Shared memory system call interface
function SHMGET(KEY : in integer;
                SIZE : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SHMGET);
pragma INTERFACE_NAME(SHMGET, "shmget");
function SHMAT(SHMID : in integer;
               SHMADDR : in system.address;
               FLAG : in integer) return system.address;
pragma INTERFACE(C, SHMAT);
pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

                d  b
                d  b
                d  b
r rrr      oooo      ddd d  b bbb
rr  r      o  o      d  dd  bb  b
r          o  o      d  d   b    b
r          o  o      d  d   b    b
r          o  o      d  dd  bb  b
r          oooo      ddd d  b bbb

```

```

cccc      oooo      m m mm      p ppp
c        c      o      o      mm m m      pp  p
c        c      o      o      m  m m      p   p
c        c      o      o      m  m m      p   p
c        c      o      o      m  m m      pp  p
cccc      oooo      m  m m      p  ppp
                        p
                        p
                        p

```

Job: rodb\_component\_data\_types\_.ada  
Date: Sun Apr 5 01:17:18 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t t
r        oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb      cccc      oooo      m m mm      p ppp
rr   r      o   o      d   dd  bb   b   c   c      o   o      mm m m      pp   p
r        o   o      d   d   b       b   c       o   o      m m m      p   p
r        o   o      d   d   b       b   c       o   o      m m m      p   p
r        o   o      d   dd  bb   b   c   c      o   o      m m m      pp   p
r        oooo      ddd d   b bbb      cccc      oooo      m m m      p ppp
                                     p
                                     p
                                     p

```

..  
..

Job: rodbcomp.dat  
Date: Sun Apr 5 01:15:23 1992



NO	Number_Of_Iterations	Times
1	1000	0.42914
2	2000	0.89508
3	3000	1.32428
4	4000	1.79968
5	5000	2.21936
6	6000	2.69482
7	7000	3.11444
8	8000	3.58990
9	9000	4.01910
10	10000	4.43866

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb  cccc  oooo  m m mm  p ppp  22222
rr  r  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p  2 2
r      o  o  d  d  b  b  c  c  o  o  m m m  p  p  2
r      o  o  d  dd  bb  b  c  c  o  o  m m m  pp  p  2
r      oooo  ddd d  b bbb  cccc  oooo  m m m  p ppp  2222222
                                           p
                                           p
                                           p

```

Job: rodbcomp2.out  
Date: Sun Apr 5 01:23:05 1992

```
put_line("Main program exception");  
end RodbtstB2;
```

```
-- This is the writing test program no semaphores just prevention of preemption
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure RodbtstB2 is
```

```
-- Constant definitions
ATTR_SIZE    : constant integer := 200;
RESULT_SIZE  : constant integer := 10;
```

```
-- Data type definition
type Result_Type is record
    Loops : integer;
    Times : duration;
end record;
```

```
-- Package instantiations
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package ROBCP renames RODB_Component;
```

```
-- Variable definition
Length          : integer;
Number_Of_Times : integer;
Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
Start_Time      : CALENDAR.time;
Finish_Time     : CALENDAR.time;
Results         : array(1..RESULT_SIZE) of Result_Type;
Outfile         : file_type;
```

```
begin
```

```
    ROBCP.Load_Comps("rodbcomp.dat");
    Length := 1;
    Addr_List(1) := 0;
    Attr_List(1) := (Type_ID => 0, Int_Value => 200);
    Number_Of_Times := 1000;
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock;
        for J in 1..Number_Of_Times loop -- One priority raise/lower per write
            ROBCP.Write_Attrs(Addr_List, Length, Attr_List);
        end loop;
        Finish_Time := CALENDAR.clock;
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;
```

```
-- Output the result to a file
create(Outfile, out_file, "rodbcomp2.out",
      form=>"world=>read, owner=>read_write");
```

```
put_line(Outfile, "NO      Number_Of_Iterations      Times");
for I in 1..RESULT_SIZE loop
    INT_IO.put(Outfile, I, width => 5);
    INT_IO.put(Outfile, Results(I).Loops);
    FIX_IO.put(Outfile, Results(I).Times);
    new_line(Outfile);
end loop;
close(Outfile);
```

```
exception
when others =>
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb      ttttt      ssss      ttttt      BBBBBB      22222
rr   r    o   o    d   dd   bb   b      t       s   s      t       B   B      2       2
r      o   o    d   d   b   b   b      t       ss      t       B   B      2       2
r      o   o    d   d   b   b   b      t       ss      t       B   B      2       2
r      o   o    d   dd   bb   b      t t      s   s      t t      B   B      2       2
r      oooo      ddd d   b bbb      tt       ssss      tt      BBBBBB      2222222

```

Job: rodbtstB2.ada  
 Date: Sun Apr 5 16:47:49 1992

NO	Number_Of_Iterations	Times
1	1000	0.48499
2	2000	0.93329
3	3000	1.36237
4	4000	1.85693
5	5000	2.28613
6	6000	2.78064
7	7000	3.21936
8	8000	3.70435
9	9000	4.14307
10	10000	4.62805

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

			d	b						1
			d	b						11
			d	b						1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp			1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p			1
r	o o	d d	b b	c	o o	m m m	p p			1
r	o o	d d	b b	c	o o	m m m	p p			1
r	o o	d dd	bb b	c c	o o	m m m	pp p			1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp			11111
							p			
							p			
							p			

Job: rodbcompl.out  
Date: Sun Apr 5 01:20:11 1992

**end RodbtstB1;**



```
-- This is the reading test program without semaphores just prevent preemption
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
procedure RodbtstB1 is
```

```
-- Constant definitions
ATTR_SIZE    : constant integer := 200;
RESULT_SIZE  : constant integer := 10;
```

```
-- Data type definition
type Result_Type is record
    Loops : integer;
    Times : duration;
end record;
```

```
-- Package instantiations
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package RODBCP renames RODB_Component;
```

```
-- Variable definition
Length          : integer;
Number_Of_Times : integer;
Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
Start_Time      : CALENDAR.time;
Finish_Time     : CALENDAR.time;
Results         : array(1..RESULT_SIZE) of Result_Type;
Outfile         : file_type;
```

```
begin
```

```
    RODBCP.Load_Comps("rodbcomp.dat");
    Length := 1;
    Addr_List(1) := 0;
    Number_Of_Times := 1000;
    for I in 1..RESULT_SIZE loop
        Start_Time := CALENDAR.clock;
        for J in 1..Number_Of_Times loop -- One priority raise/lower per read
            RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
        end loop;
        Finish_Time := CALENDAR.clock;
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
        Number_Of_Times := Number_Of_Times + 1000;
    end loop;
```

```
-- Output the result to a file
create(Outfile, out_file, "rodbcomp1.out",
    form=>"world=>read, owner=>read_write");
put_line(Outfile, "NO      Number_Of_Iterations    Times");
for I in 1..RESULT_SIZE loop
    INT_IO.put(Outfile, I, width => 5);
    INT_IO.put(Outfile, Results(I).Loops);
    FIX_IO.put(Outfile, Results(I).Times);
    new_line(Outfile);
end loop;
close(Outfile);
```

```
exception
    when others =>
        put_line("Main program exception");
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r       o   o    o   o    t
r       o   o    o   o    t
r       o   o    o   o    t t
r       oooo      oooo      tt

```

```

                        d   b
                        d   b
                        d   b
r rrr      oooo      ddd d   b bbb      t      t      BBBBBB      1
rr   r    o   o    d   dd   bb   b      t      t      B   B      11
r       o   o    d   d   b   b   b      t      t      B   B      1 1
r       o   o    d   d   b   b   b      t      t      B   B      1
r       o   o    d   dd   bb   b      t      t      B   B      1
r       oooo      ddd d   b bbb      tt     ssss      tt     BBBBBB      11111

```

Job: rodbtstB1.ad  
 Date: Sun Apr 5 16:46:38 1992

This is directory for Appendix C-4

It contains all the programs for the "realistic" use of prevention of preemption where priority is raised and lowered just once for each read or write.

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
r

  oooo      oooo
o   o      o   o
o   o      o   o
o   o      o   o
o   o      o   o
  oooo      oooo

```

```

      d
      d
      d
    ddd d
d   dd
d   d
d   d
d   dd
ddd d

      i
      ii
      i
      i
      i
      i
      iii

r rrr
rr  r
r
r
r
r

  eeee      e
e   e
e       e
e       e
e       e
  eeee

  cccc      c
c   c
c   c
c   c
c   c
  cccc

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      ..
      ..

      x   x
      x   x
      xx
      xx
      x   x
      x   x

```

Job: direct.txt  
Date: Sun Apr 5 16:49:52 1992

## **Appendix C-4**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**Protected by Disabling Preemption before each Read or Write and Re-enabling After.**

\_\_\_\_\_

```

/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */ -
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}

```

—



r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
	oooo	oooo	tt

ssss	eeee	m m mm	ssss	r rrr	m m mm	v v		cccc
s s	e e	mm m m	s s	rr r	mm m m	v v		c c
ss	eeeeee	m m m	ss	r	m m m	v v		c
ss	e	m m m	ss	r	m m m	v v		c
s s	e e	m m m	s s	r	m m m	v v	..	c c
ssss	eeee	m m m	ssss	r	m m m	v	..	cccc

Job: semsrmv.c  
Date: Thu Apr 9 13:20:36 1992

```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*  
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/sem.h>  
int semsinit(semid)  
    int semid;  
{  
    short initarray[3];  
    int    flag;  
    void    perror();  
  
    initarray[0] = initarray[1] = initarray[2] = 0;  
    flag = semctl(semid, 3, SETALL, initarray);  
    if (flag == -1) {  
        perror("semsinit fails: ");  
    }  
    return(flag);  
}
```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

ssss	eeee	m m mm	ssss	i		i	t	
s s	e e	mm m m	s s	ii	n nnn	ii	t	
ss	eeeeee	m m m	ss	i	nn n	i	ttttt	
ss	e	m m m	ss	i	n n	i	t	
s s	e e	m m m	s s	i	n n	i	t	
ssss	eeee	m m m	ssss	iii	n n	iii	t t	..
							tt	..

Job: semsinit.c  
Date: Thu Apr 9 13:20:21 1992

```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */ /* Commented out for this test */
    /* sole_writer[0] = WREAD_END; */ /* Allow writer in */
    /* sole_writer[1] = WWRITE_UNLOCK; */ /* Allow reader in */
    /* flag = semop(semid, sole_writer, 2); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("writeend fails: ");
    } */

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

                                     d
                                     d
                                     d
                                     d
w      w  r rrr  ii  t  ttttt  eeee  eeee  n nnn  ddd d
w w w rr  r  i  t  e  e  nn  n  d  dd
w w w r  i  t  e  e  n  n  d  d
w w w r  i  t  e  e  n  n  d  dd
w w w r  i  t  t  e  e  n  n  ddd d  ..
  ww ww r  iii  tt  eeee  eeee  n  n

```

Job: writeend.c  
Date: Thu Apr 9 13:20:16 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START     = { 0, 1, 0};
struct sembuf WWRITE_LOCK     = { 1, 1, 0};
struct sembuf WWRITE_DESIRE    = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Make write request by doing a semaphore operation */ /*Commented out*/
    /* flag = semop(semid, &WWRITE_DESIRE, 1); */
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    } /*

    /* Perform four semaphore operations */ /* Commented out for this test */
    /* sole_writer[0] = WAIT_NO_READERS; */ /* Wait for no more readers */
    /* sole_writer[1] = WWRITE_LOCK; */ /* preventing succeeding readers */
    /* sole_writer[2] = WREAD_START; */ /* preventing succeeding writers */
    /* sole_writer[3] = WIN_PROGRESS_WRITE; */ /* Cancel the write-request */
    /* flag = semop(semid, sole_writer, 4); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    } */

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      oooo  oooo  tt

```

```

      i      t      b
      t      b
      t      b
w      w  r rrr  ii  ttttt  eeee  b bbb  eeee  ggg g
w  w  w  rr  r  i  t  e  e  bb  b  e  e  g  gg
w  w  w  r  i  t  eeeee  b  b  eeeee  g  g
w  w  w  r  i  t  e  b  b  e  g  g
w  w  w  r  i  t  t  e  e  bb  b  e  e  g  gg
ww ww  r  iii  tt  eeee  b bbb  eeee  ggg g
      g
      g
      gggg
      ..
      ..

```

Job: writebeg.c  
Date: Thu Apr 9 13:20:04 1992

```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */ /* commented out for this test */
    /* flag = semop(semid, &RREAD_END, 1); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("readend fails: ");
    } */

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```



```

      t
      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o      o      o      t
r      o      o      o      t
r      o      o      o      t
r      o      o      o      t
r      oooo      oooo      tt

```

```

      d      d      d
      d      d      d
      d      d      d
r rrr      eeee      aaaa      ddd d      eeee      n nnn      ddd d      cccc
rr   r      e   e      a      d   dd      e   e      nn   n      d   dd      c   c
r      eeeee      aaaaa      d   d      eeeee      n   n      d   d      c
r      e      a      a      d   d      e      n   n      d   d      c
r      e   e      a      aa      d   dd      e   e      n   n      d   dd      ..      c   c
r      eeee      aaaa a      ddd d      eeee      n   n      ddd d      ..      cccc

```

Job: readend.c  
Date: Thu Apr 9 13:19:57 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t    my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform three semaphore operations */ /* Commented out for this test */
    /* one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[4] = RREAD_START; */ /* Prevent writer in */
    /* flag = semop(semid, one_of_n_readers, 5); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("readbeg fails: ");
    } */

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t
r      o   o    o   o    t t
r      oooo      oooo      tt

```

```

                        d   b
                        d   b
                        d   b
r rrr      eeee      aaaa      ddd d   b bbb      eeee      ggg g      cccc
rr   r    e   e      a      d   dd   bb   b   e   e      g   gg      c   c
r      eeeee      aaaaa      d   d   b   b   eeeee      g   g      c
r      e      a      a      d   d   b   b   e      g   g      c
r      e   e      a      aa      d   dd   bb   b   e   e      g   gg      ..   c
r      eeee      aaaa a      ddd d   b bbb      eeee      ggg g      ..   ccc
                                           g
                                           g
                                           gggg

```

Job: readbeg.c  
 Date: Thu Apr 9 13:19:52 1992

```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is
    Length : Integer;
    Temp_Pos : integer;
    Temp_Flt : float;
    Temp_Addr : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

```

pragma page;

```

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;

```

```

begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

```

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;

```

```

begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```

pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

Length : Integer;  
Temp\_Pos : integer;  
Temp\_Int : integer;  
Temp\_Addr : system.address;

begin

INT\_IO.get(Infile, Length);

skip\_line(Infile);

for I in 1..Length loop

INT\_IO.get(Infile, Temp\_Pos);

INT\_IO.get(Infile, Temp\_Int);

skip\_line(Infile);

if (Temp\_Pos < 0) or (Temp\_Pos > RCDT.INT\_SIZE-1) then  
raise Shm\_Outrange;

end if;

Temp\_Addr := Shmaddr + system.offset(Temp\_Pos\*4);

RCDT.AINT(Temp\_Addr, Temp\_Int);

end loop;

end Load\_Ints;

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

Length : Integer;  
Temp\_Pos : integer;  
Temp\_Char : character;  
Temp\_Addr : system.address;

begin

INT\_IO.get(Infile, Length);

skip\_line(Infile);

for I in 1..Length loop

INT\_IO.get(Infile, Temp\_Pos);

get(Infile, Temp\_Char);

-- Skip a space

get(Infile, Temp\_Char);

skip\_line(Infile);

if (Temp\_Pos < 0) or (Temp\_Pos > RCDT.CHAR\_SIZE-1) then  
raise Shm\_Outrange;

end if;

Temp\_Addr := Shmaddr + system.offset(RCDT.CHAR\_OFFSET+Temp\_Pos);

RCDT.ACHAR(Temp\_Addr, Temp\_Char);

end loop;

end Load\_Chars;

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

Length : Integer;  
Temp\_Pos : integer;  
Temp\_Bool : boolean;  
Temp\_Addr : system.address;

begin

INT\_IO.get(Infile, Length);

skip\_line(Infile);

for I in 1..Length loop

INT\_IO.get(Infile, Temp\_Pos);

BOOL\_IO.get(Infile, Temp\_Bool);

skip\_line(Infile);

if (Temp\_Pos < 0) or (Temp\_Pos > RCDT.BOOL\_SIZE-1) then

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
          form=>"world=>read, owner=>read_write");

    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_BoolS(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_BoolS(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```

```

--
...
procedure Load_Comps(Filename : in string) is
  Infile : FILE_TYPE;
  Temp : system.address;
  Flag : integer;
begin
  open(Infile, in_file, Filename);

  -- Initialize RODB Integer Component
  for I in 1..RCDT.INT_SIZE loop
    Temp := Shmaddr + system.offset((I-1)*4);
    RCDT.AINT(Temp, 0);
  end loop;
  Load_Ints(Infile);

  -- Initialize RODB Character Component
  for I in 1..RCDT.CHAR_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
  end loop;
  Load_Chars(Infile);

  -- Initialize RODB Boolean Component
  for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
  end loop;
  Load_Bools(Infile);

  -- Initialize RODB Float Component
  for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
  end loop;
  Load_Flts(Infile);

  close(Infile);

  Flag := RCDT.SEMSINIT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;

exception
  when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
  when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
  when Sem_Exception =>
    put_line("Semaphore cannot be initialized.");
    raise Sem_Exception;
  when others =>
    put_line("Unknown exception.");
    put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```



```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```
/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```

## **Appendix C-5**

### **Uncontested RODB Reads and Uncontested RODB Writes**

**No disabling of preemption  
No Semaphore Protection  
Just "raw" RODB Reads and Writes**



```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d
      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr  r  e  e  a  d  dd
r      eeeee  aaaaa  d  d
r      e  a  a  d  d
r      e  e  a  aa  d  dd
r      eeee  aaaa a  ddd d

```

```

m m mm  eeee
mm m m  e  e
m m m  eeeee
m m m  e
.. m m m  e  e
.. m m m  eeee

```

Job: read.me  
Date: Sun Apr 5 20:20:45 1992

This directory stores all the files to build up RODB "attribute" components. The protection mechanism is absent for this test. There is no mechanism used to assure mutual exclusion. Both the prevention-of-preemption by raising process priority and the semaphore operations were disabled. WHAT WE ARE TRYING TO MEASURE IS THE TIME REQUIRED FOR JUST THE "RAW" READS OR WRITES. THE TEST LOOPS TO MEASURE HOW LONG IT TAKES FOR 1000 to 10000 READS OR WRITES. THE RESULTS FOR READING ARE IN FILE rodbcomp1.out and the RESULTS FOR WRITING ARE IN FILE rodbcomp2.out. The reads and writes are non-competing.

```

      t
      t
r rrr      oooo      oooo      ttttt
rr      r  o      o  o      o      t
r          o      o  o      o      t
r          o      o  o      o      t
r          o      o  o      t  t
r          oooo      oooo      tt

```

```

                        d      b
                        d      b
                        d      b
r rrr      oooo      ddd d      b bbb      ttttt      ssss      ttttt      C      C      1
rr      r  o      o  d      dd      bb      b      t      s      s      t      C      C      11
r          o      o  d      d      b      b      t      s      s      t      C      C      1 1
r          o      o  d      d      b      b      t      s      s      t      C      C      1 1
r          o      o  d      dd      bb      b      t      s      s      t      C      C      1
r          oooo      ddd d      b bbb      tt      ssss      tt      CCCC      11111

```

Job: rodbtstC1.ada  
Date: Sun Apr 5 20:27:34 1992

```

-- This is the reading test program without semaphores or preemption prevention
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types_Mod, RODB_Component_Mod
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types_Mod, RODB_Component_Mod
procedure RodbtstC1 is
  -- this program is a mod of Rodbtst81 to eliminate all protection
  -- to the shared memory just to find out the time to do shared mem reads
  -- Constant definitions
  ATTR_SIZE    : constant integer := 200;
  RESULT_SIZE  : constant integer := 10;

  -- Data type definition
  type Result_Type is record
    Loops : integer;
    Times : duration;
  end record;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package FIX_IO is new TEXT_IO.FIXED_IO(duration);
  package RCDT renames RODB_Component_Data_Types_Mod;
  package RODBCP renames RODB_Component_Mod;

  -- Variable definition
  Length          : integer;
  Number_Of_Times : integer;
  Addr_List       : RCDT.Pos_List_Type(1..ATTR_SIZE);
  Attr_List       : RCDT.Attr_List_Type(1..ATTR_SIZE);
  Start_Time      : CALENDAR.time;
  Finish_Time     : CALENDAR.time;
  Results         : array(1..RESULT_SIZE) of Result_Type;
  Outfile         : file_type;

begin
  RODBCP.Load_Comps("rodbcomp.dat");
  Length := 1;
  Addr_List(1) := 0;
  Number_Of_Times := 1000;
  for I in 1..RESULT_SIZE loop
    Start_Time := CALENDAR.clock;
    for J in 1..Number_Of_Times loop -- no priority raise/lower no semaphores
      RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
    end loop;
    Finish_Time := CALENDAR.clock;
    Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
    Number_Of_Times := Number_Of_Times + 1000;
  end loop;

  -- Output the result to a file
  create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
  put_line(Outfile, "NO      Number_Of_Iterations    Times");
  for I in 1..RESULT_SIZE loop
    INT_IO.put(Outfile, I, width => 5);
    INT_IO.put(Outfile, Results(I).Loops);
    FIX_IO.put(Outfile, Results(I).Times);
    new_line(Outfile);
  end loop;
  close(Outfile);
exception
  when others =>

```



```
put_line("Main program exception");  
end RodbtstC1;
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t t
r      oooo      oooo      tt

```

```

                        d   b
                        d   b
                        d   b
r rrr      oooo      ddd d   b bbb      t      t      CCCC      22222
rr   r      o   o      d   dd  bb   b      ttttt  ssss      ttttt  C   C   2
r        o   o      d   d   b   b      t   s   s      t   C   C   2
r        o   o      d   d   b   b      t   ss      t   C   C   2
r        o   o      d   dd  bb   b      t t   s   s      t t  C   C   2
r      oooo      ddd d   b bbb      tt      ssss      tt      CCCC  222222

```

Job: rodbtstC2.ada  
Date: Sun Apr 5 20:27:41 1992

```
-- This is the writing test program with no Preemption Control, no semaphores
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types_Mod, RODB_Component_Mod;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types_Mod, RODB_Component_Mod;
procedure RodbtstC2 is
```

```
-- This program tests uncontested writes to shared memory
```

```
-- Constant definitions
```

```
ATTR_SIZE : constant integer := 200;
```

```
RESULT_SIZE : constant integer := 10;
```

```
-- Data type definition
```

```
type Result_Type is record
```

```
    Loops : integer;
```

```
    Times : duration;
```

```
end record;
```

```
-- Package instantiation
```

```
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
```

```
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
```

```
package RCDT renames RODB_Component_Data_Types_Mod;
```

```
package RODBCP renames RODB_Component_Mod;
```

```
-- Variable definition
```

```
Length : integer;
```

```
Number_Of_Times : integer;
```

```
Addr_List : RCDT.Pos_List_Type(1..ATTR_SIZE);
```

```
Attr_List : RCDT.Attr_List_Type(1..ATTR_SIZE);
```

```
Start_Time : CALENDAR.time;
```

```
Finish_Time : CALENDAR.time;
```

```
Results : array(1..RESULT_SIZE) of Result_Type;
```

```
Outfile : file_type;
```

```
begin
```

```
    RODBCP.Load_Comps("rodbcomp.dat");
```

```
    Length := 1;
```

```
    Addr_List(1) := 0;
```

```
    Attr_List(1) := (Type_ID => 0, Int_Value => 200);
```

```
    Number_Of_Times := 1000;
```

```
    for I in 1..RESULT_SIZE loop
```

```
        Start_Time := CALENDAR.clock;
```

```
        for J in 1..Number_Of_Times loop -- No priority raise/lower no semaphores
```

```
            RODBCP.Write_Attrs(Addr_List, Length, Attr_List);
```

```
        end loop;
```

```
        Finish_Time := CALENDAR.clock;
```

```
        Results(I) := (Number_Of_Times, Finish_Time-Start_Time);
```

```
        Number_Of_Times := Number_Of_Times + 1000;
```

```
    end loop;
```

```
-- Output the result to a file
```

```
create(Outfile, out_file, "rodbcomp2.out",
      form=>"world=>read, owner=>read_write");
```

```
put_line(Outfile, "NO      Number_Of_Iterations      Times");
```

```
for I in 1..RESULT_SIZE loop
```

```
    INT_IO.put(Outfile, I, width => 5);
```

```
    INT_IO.put(Outfile, Results(I).Loops);
```

```
    FIX_IO.put(Outfile, Results(I).Times);
```

```
    new_line(Outfile);
```

```
end loop;
```

```
close(Outfile);
```

```
exception
```

```
when others =>
```

```
    put_line("Main program exception");  
end RodbtstC2;
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t
r        o   o      o   o      t t
r        oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb      cccc      oooo      m m mm      p ppp
rr   r      o   o      d   dd   bb   b   c   c      o   o      mm m m      pp   p
r        o   o      d   d   b       b   c       o   o      m m m      p   p
r        o   o      d   d   b       b   c       o   o      m m m      p   p
r        o   o      d   dd   bb   b   c   c      o   o      m m m      pp   p
r        oooo      ddd d   b bbb      cccc      oooo      m m m      p ppp
                                           p
                                           p
                                           p

```

Job: rodbcomp.dat  
Date: Sun Apr 5 20:27:54 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	tttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

		d	b						1
		d	b						11
		d	b						1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp		1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d d	b b	c	o o	m m m	p p		1
r	o o	d dd	bb b	c c	o o	m m m	pp p		1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp		11111
							p		
							p		
							p		

Job: rodbcompl.out  
Date: Sun Apr 5 20:24:50 1992

NO	Number_Of_Iterations	Times
1	1000	0.08588
2	2000	0.15259
3	3000	0.23840
4	4000	0.37054
5	5000	0.39105
6	6000	0.53265
7	7000	0.55316
8	8000	0.68530
9	9000	0.76154
10	10000	0.83789



```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          oooo      oooo      tt

```

```

                        d      b
                        d      b
                        d      b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp      22222
rr  r      o      o      d      dd      bb      b      c      c      o      o      mm m m      pp      p      2      2
r          o      o      d      d      b      b      c      c      o      o      m m m      p      p      2
r          o      o      d      dd      bb      b      c      c      o      o      m m m      pp      p      2
r          oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp      2222222

```

Job: rodbcomp2.out  
 Date: Sun Apr 5 20:25:01 1992

NO	Number_Of_Iterations	Times
1	1000	0.07629
2	2000	0.16211
3	3000	0.23840
4	4000	0.37061
5	5000	0.39099
6	6000	0.52313
7	7000	0.56268
8	8000	0.68530
9	9000	0.76160
10	10000	0.83783

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r       o   o    o   o    t
r       o   o    o   o    t
r       o   o    o   o    t t
r       oooo      oooo      tt

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb
rr   r    o   o    d   dd  bb   b
r       o   o    d   d   b     b
r       o   o    d   d   b     b
r       o   o    d   dd  bb   b
r       oooo      ddd d   b bbb

```

```

      cccc      oooo      m m mm      p ppp
c     c    o   o    mm m m      pp   p
c     c    o   o    m m m      p    p
c     c    o   o    m m m      p    p
c     c    o   o    m m m      pp   p
cccc      oooo      m m m      p ppp
                               p
                               p
                               p

```

---

Job: rodb\_component\_data\_types\_mod\_.ada  
 Date: Sun Apr 5 20:23:06 1992

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
-- In this case we do not need to make any C calls so they have been commented
-- out. This is system rodbtstb1
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES_MOD is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY : in integer;
                 SIZE : in integer;
                 FLAG : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE(NAME(SHMGET, "shmget"));
  function SHMAT(SHMID : in integer;
                SHMADDR : in system.address;
                FLAG : in integer) return system.address;

```

```

pragma INTERFACE(C, SHMAT);
pragma INTERFACE_NAME(SHMAT, "shmat");
function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

```

-- Semaphore system call and C function interface -- no semaphores here
-- function SEMGET(KEY : in integer;
--                NSEMS : in integer;
--                FLAG : in integer) return integer;
-- pragma INTERFACE(C, SEMGET);
-- pragma INTERFACE_NAME(SEMGET, "semget");
-- function SEMSINIT(SEMID : in integer) return integer;
-- pragma INTERFACE(C, SEMSINIT);
-- pragma INTERFACE_NAME(SEMSINIT, "semsinit");
-- function SEMPRINT(SEMID : in integer) return integer;
-- pragma INTERFACE(C, SEMPRINT);
-- pragma INTERFACE_NAME(SEMPRINT, "semprint");
-- function READBEG(SEMID : in integer) return integer;
-- pragma INTERFACE(C, READBEG); -- no readbeg.c
-- pragma INTERFACE_NAME(READBEG, "readbeg");
-- function READEND(SEMID : in integer) return integer;
-- pragma INTERFACE(C, READEND); -- no readend.c
-- pragma INTERFACE_NAME(READEND, "readend");
-- function WRITEBEG(SEMID : in integer) return integer;
-- pragma INTERFACE(C, WRITEBEG);
-- pragma INTERFACE_NAME(WRITEBEG, "writebeg"); -- no writebeg.c
-- function WRITEEND(SEMID : in integer) return integer;
-- pragma INTERFACE(C, WRITEEND);
-- pragma INTERFACE_NAME(WRITEEND, "writeend"); -- no writend.c either
-- function SEMSRMV(SEMID : in integer) return integer;
-- pragma INTERFACE(C, SEMSRMV);
-- pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

```

end RODB_Component_Data_Types_Mod;

```

				d	b
				d	b
				d	b
r rrr	oooo	ddd d		b bbb	
rr r	o o	d dd		bb b	
r	o o	d d		b b	
r	o o	d d		b b	
r	o o	d dd		bb b	
r	oooo	ddd d		b bbb	

Job: rodb\_component\_mod .ada  
Date: Sun Apr 5 20:22:11 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types_Mod;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types_Mod;
package Rodb_Component_Mod is

  -- Package renaming
  package RCDT renames Rodb_Component_Data_Types_Mod;

  -- Exception definition
  Shm_Exception : exception;
  Shm_Outrange  : exception;
  Sem_Exception : exception;

  -- Read attributes from RODB components
  procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                      Length      : in      integer;
                      Attr_List   : in out RCDT.Attr_List_Type);

  -- Write attributes to RODB components
  procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                      Length      : in      integer;
                      Attr_List   : in      RCDT.Attr_List_Type);

  -- Print out the semaphore values -- don't need this now
  -- procedure Print_Sems;

  -- Load RODB components from a disk file
  procedure Load_Comps(Filename : in      string);

  -- Save RODB components to a disk file
  procedure Save_Comps(Filename : in      string);

  -- Shutdown the RODB components
  procedure Shutdown_Comps;

end Rodb_Component_Mod;

```

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
	oooo	oooo	tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
	oooo	ddd d	b bbb

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp p
c	o o	m m m	p p
c	o o	m m m	pp p
c c	o o	m m m	pp ppp
cccc	oooo	m m m	p ppp
			p
			p
			p

Job: rodb\_component\_mod.ada  
Date: Sun Apr 5 20:22:25 1992



```

with TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types_Mod;
use TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types_Mod;
package body Rodb_Component_Mod is

```

```

-- In this program the calls to readbeg.c and readend.c were eliminated,
-- the reason being that no semaphores or dynamic priority operations
-- were used. The concept was to see what time it would take for 1000,
-- 2000,.. 5000, etc. unencumbered reads and writes.

```

```

--Local variables
Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer := 0;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    -- Flag := RCDT.READBEG(Semid); -- don't need readbeg if no semaphore
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    -- if Flag = -1 then
    --     raise Sem_Exception;
    -- end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    -- Flag := RCDT.READEND(Semid); -- don't need readend if no semaphore
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    -- if Flag = -1 then
    --     raise Sem_Exception;
    -- end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components
procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                      Length      : in      integer;
                      Attr_List   : in      RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer:= 0;
begin
    -- PREEMPTION_CONTROL.DISABLE PREEMPTION;
    -- Flag := RCDT.WRITEBEG(Semid); -- Don't need if no semaphores
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    -- if Flag = -1 then
    --     raise Sem_Exception;
    -- end if;
    for I in 1.. Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            RCDT.AINT(Temp, Attr_List(I).Int_Value);
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
        else
            RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE PREEMPTION;
--    Flag := RCDT.WRITEEND(Semid); -- don't need for no semaphore
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    -- if Flag = -1 then
    --     raise Sem_Exception;
    -- end if;
end Write_Attrs;

-- Print out the semaphore values -- but now there aren't any

-- procedure Print_Sems is
--     Flag : integer;
--     begin
--         Flag := RCDT.SEMPRINT(Semid);
--         if Flag = -1 then
--             raise Sem_Exception;
--         end if;
--     end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--     Number_Of_Integers
--         Position1   Integer1
--         Position2   Integer2
--     ...
--     Number_Of_Characters
--         Position1   Character1
--         Position2   Character2
--     ...
--     Number_Of_Booleans

```

```

--      Position1  Boolean1
--      Position2  Boolean2
--      ...
--      Number_Of_Floats
--      Position1  Float1
--      Position2  Float2
--      ...
procedure Load_Comps(Filename : in      string) is
  Infile : FILE_TYPE;
  Temp   : system.address;
  Flag   : integer;
begin
  open(Infile, in_file, Filename);

  -- Initialize RODB Integer Component
  for I in 1..RCDT.INT_SIZE loop
    Temp := Shmaddr + system.offset((I-1)*4);
    RCDT.AINT(Temp, 0);
  end loop;
  Load_Ints(Infile);

  -- Initialize RODB Character Component
  for I in 1..RCDT.CHAR_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
  end loop;
  Load_Chars(Infile);

  -- Initialize RODB Boolean Component
  for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
  end loop;
  Load_Bools(Infile);

  -- Initialize RODB Float Component
  for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
  end loop;
  Load_Flts(Infile);

  close(Infile);

  --  Flag := RCDT.SEMSINIT(Semid);
  --  if Flag = -1 then
  --    raise Sem_Exception;
  --  end if;

  exception
  when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
  when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
  --  when Sem_Exception =>
  --    put_line("Semaphore cannot be initialized.");
  --    raise Sem_Exception;

```

```

    when others =>
        put_line("Unknown exception.");
        put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file
-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
    Outfile : FILE_TYPE;
begin
    if Filename /= "" then
        create(Outfile, out_file, Filename,
              form=>"world=>read, owner=>read_write");

        Save_Ints(Outfile);
        Save_Chars(Outfile);
        Save_Bools(Outfile);
        Save_Flts(Outfile);
        close(Outfile);
    else
        Save_Ints(TEXT_IO.standard_output);
        Save_Chars(TEXT_IO.standard_output);
        Save_Bools(TEXT_IO.standard_output);
        Save_Flts(TEXT_IO.standard_output);
    end if;
exception
    when constraint_error =>
        put_line("RODB Components data collapsed.");
        put_line("Saving components fails!");
    when others =>
        put_line("Unknown exception.");
        put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
    Flag : integer;
begin
    Flag := RCDT.SHMDT(Shmaddr);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
    Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
    if Flag = -1 then
        raise Shm_Exception;
    end if;
end Shutdown_Comps;

```

```

    end if;
    -- Flag := RCDT.SEMSRMV(Semid);
    -- if Flag = -1 then
    --     raise Sem_Exception;
    -- end if;
end Shutdown_Comps;

pragma page;

-- Load all the integers from a disk file to RODB Integer Component
procedure Load_Ints(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Int    : integer;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        INT_IO.get(Infile, Temp_Int);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
        RCDT.AINT(Temp_Addr, Temp_Int);
    end loop;
end Load_Ints;

-- Load all the characters from a disk file to RODB Character Component
procedure Load_Chars(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Char   : character;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        get(Infile, Temp_Char);           -- Skip a space
        get(Infile, Temp_Char);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
        RCDT.ACHAR(Temp_Addr, Temp_Char);
    end loop;
end Load_Chars;

-- Load all the booleans from a disk file to RODB Boolean Component
procedure Load_Bools(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Bool   : boolean;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);

```

```

skip_line(Infile);
for I in 1..Length loop
  INT_IO.get(Infile, Temp_Pos);
  BOOL_IO.get(Infile, Temp_Bool);
  skip_line(Infile);
  if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
    raise Shm_Outrange;
  end if;
  Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
  RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is
  Length : Integer;
  Temp_Pos : integer;
  Temp_Flt : float;
  Temp_Addr : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    FLT_IO.get(Infile, Temp_Flt);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
    RCDT.AFLT(Temp_Addr, Temp_Flt);
  end loop;
end Load_Flts;

pragma page;

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in FILE_TYPE) is
  Temp_Addr : system.address;
begin
  put(Outfile, "Number Of Integers is: ");
  Int_IO.put(Outfile, RCDT.INT_SIZE);
  new_line(Outfile);
  for I in 1..RCDT.INT_SIZE loop
    put(Outfile, "Integer number ");
    Int_IO.put(Outfile, I-1, width => 5);
    put(Outfile, ":");
    Temp_Addr := Shmaddr + system.offset((I-1)*4);
    Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
    new_line(Outfile);
  end loop;
end Save_Ints;

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in FILE_TYPE) is
  Temp_Addr : system.address;
begin
  put(Outfile, "Number Of Characters is: ");
  Int_IO.put(Outfile, RCDT.CHAR_SIZE);
  new_line(Outfile);

```

```

    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    -- Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    -- if Semid = -1 then
    --     raise Sem_Exception;
    -- end if;
end Rodb_Component_Mod;

```





```

      t
      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  eeee  aaaa  ddd d  b bbb  eeee  ggg g  cccc
rr   r  e   e  a   a  d  dd  bb  b  e   e  g  gg  c   c
r      eeeee  aaaaa  d  d  b   b  eeeee  g  g  c
r      e   e  a   a  d  d  b   b  e   e  g  g  c
r      e   e  a   aa  d  dd  bb  b  e   e  g  gg  ..  c   c
r      eeee  aaaa a  ddd d  b bbb  eeee  ggg g  ..  cccc

      g
      g
      gggg

```

Job: readbeg.c  
Date: Sun Apr 5 20:23:31 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
/* for this test this program has been modified to do nothing */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */ /* No priority operations here */

    /* Perform three semaphore operations */ /* Also no semops here */
    /* one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[4] = RREAD_START; */ /* Prevent writer in */
    /* flag = semop(semid, one_of_n_readers, 5); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("readbeg fails: ");
    } */

    /* Lower the priority to the normal */
    /* fast_setprio(my_tid, my_prio); */ /* No priority operations for this test */

    return 0;
}

```

r rrr	oooo	oooo	t t t t t
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

			d d d d			d d d d	
r rrr	eeee	aaaa	ddd d	eeee	n nnn	ddd d	cccc
rr r	e e	a	d dd	e e	nn n	d dd	c c
r	eeeeee	aaaaa	d d	eeeeee	n n	d d	c
r	e	a a	d d	e	n n	d d	c
r	e e	a aa	d dd	e e	n n	d dd	.. c
r	eeee	aaaa a	ddd d	eeee	n n	ddd d	.. cccc

Job: readend.c  
Date: Sun Apr 5 20:23:38 1992

```

/* File: readend.c This is read end subroutine to reset protection */
/* for this test this program has been modified to do nothing */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t  my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */ /* No priority ops */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */ /* Commented out */

    /* Perform the semaphore operation */ /* No semops either */
    /* flag = semop(semid, &RREAD_END, 1); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("readend fails: ");
    } */

    /* Lower the priority to the normal */
    /* fast_setprio(my_tid, my_prio); */ /* Commented out */

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r        oooo      oooo      tt

```

```

      i      t      b
      t      b
      t      b
w      w  r rrr      ii      ttttt      eeee      b bbb      eeee      ggg g
w w w rr   r      i      t      e   e      bb   b      e   e      g   gg
w w w r        i      t      eeeee      b   b      eeeee      g   g
w w w r        i      t      e       b   b      e       g   g
w w w r        i      t t      e   e      bb   b      e   e      g   gg
ww ww r      iii      tt      eeee      b bbb      eeee      ggg g
                                g
                                g
                                gggg
..
..

```

Job: writebeg.c  
Date: Sun Apr 5 20:23:47 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
/* in this directory this program has been modified to do nothing */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0};
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE     = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    /* my_tid = gettid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31); */ /* Eliminate priority operations */

    /* Make write request by doing a semaphore operation */
    /* flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    } */

    /* Perform four semaphore operations */ /* Semops also eliminated */
    /* sole_writer[0] = WWAIT_NO_READERS; */ /* Wait for no more readers */
    /* sole_writer[1] = WWRITE_LOCK; */ /* preventing succeeding readers */
    /* sole_writer[2] = WREAD_START; */ /* preventing succeeding writers */
    /* sole_writer[3] = WIN_PROGRESS_WRITE; */ /* Cancel the write-request */
    /* flag = semop(semid, sole_writer, 4); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    } */

    /* Lower the priority to the normal */
    /* fast_setprio(my_tid, my_prio); */ /* no priority operations */

    return 0;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o      o      o      t
r          o      o      o      t
r          o      o      o      t
r          o      o      t      t
r          oooo      oooo      tt

```

```

                                     d
                                     d
                                     d
      i          t          ttttt      eeee      eeee      n nnn      ddd d
w      w      r rrr      ii          t          e   e      e   e      nn   n      d   dd
w w w      rr  r          i          t          eeeee      eeeee      n   n      d   d
w w w      r          i          t          e          e          n   n      d   d
w w w      r          i          t t          e   e      e   e      n   n      d   dd
w w w      r          iii         tt          eeee      eeee      n   n      ddd d
ww ww      r

```

Job: writeend.c  
 Date: Sun Apr 5 20:23:56 1992

```

/* File: writeend.c This is write end subroutine to reset write protection */
/* in this directory this program has been modified to do nothing */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    /* my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);
    */ /* no priority operations for this test */
    /* Perform the semaphore operation */ /* No semaphore ops either */
    /* sole_writer[0] = WREAD_END; */ /* Allow writer in */
    /* sole_writer[1] = WWRITE_UNLOCK; */ /* Allow reader in */
    /* flag = semop(semid, sole_writer, 2); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("writeend fails: ");
    } */

    /* Lower the priority to the normal */
    /* fast_setprio(my_tid, my_prio); */ /* Don't lower what hasn't been raised */

    return flag;
}

```



i		i	t	
			t	
ii	n nnn	ii	ttttt	
i	nn n	i	t	
i	n n	i	t	
i	n n	i	t	
i	n n	i	t t	..
iii	n n	iii	tt	..

```
Job:  semsinit.c
Date:  Sun Apr  5 20:24:10 1992
```

```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*,
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

                                     i
                                     t
                                     t
ssss  eeee  m m mm  p ppp  r rrr  ii  n nnn  ttttt
s    s  e    e  mm m m  pp  p  rr  r  i  nn  n  t
ss    eeeee  m m m m  p  p  r      i  n  n  t
  ss    e    e  m m m m  p  p  r      i  n  n  t
s    s  e    e  m m m m  pp  p  r      i  n  n  t  t  ..
ssss  eeee  m m m m  p ppp  r      iii  n  n  tt  ..

p
p
p

```

Job: semprint.c  
Date: Sun Apr 5 20:24:21 1992

```

/*File:semprint.c This is semaphore print subroutine to print semaphore values*
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semprint(semid)
    int semid;
{
    short outarray[3];
    int    flag;
    void    perror();
    int    i;

    flag = semctl(semid, 3, GETALL, outarray);
    if (flag == -1) {
        perror("semprint fails: ");
    }
    for (i=0; i<3; ++i) {
        printf("Semaphore %d has the value of %d\n", i, outarray[i]);
    }
    return(flag);
}

```

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

ssss	eeee	m m mm	ssss	r rrr	m m mm	v v		cccc
s s	e e	mm m m	s s	rr r	mm m m	v v		c c
ss	eeeeee	m m m	ss	r	m m m	v v		c
ss	e	m m m	ss	r	m m m	v v		c
s s	e e	m m m	s s	r	m m m	v v	..	c c
ssss	eeee	m m m	ssss	r	m m m	v	..	cccc

Job: sensrmv.c  
Date: Sun Apr 5 20:24:38 1992

```
/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```



---

Copies of this publication have been deposited with the Texas State Library in compliance with the State Depository Law.

---



# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

**Appendices D1 - D3 & E**

*Prepared for*

***NASA/JSC Data Management Systems  
for Space Station Freedom (SSF)***

Prepared by Co-Principal Investigators:

**T. F. Leibfried Jr.  
Sadegh Davari**  
*University of Houston-Clear Lake*

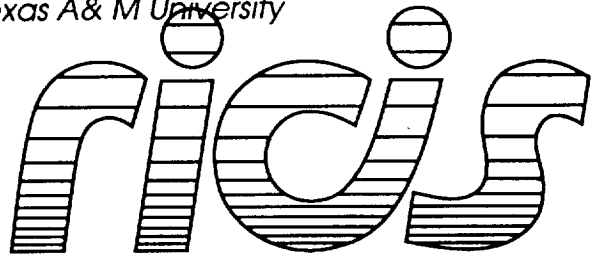
**Swami Natarajan  
Wei Zhao**  
*Texas A&M University*

Research Associates:

**Libin Wu**  
*University of Houston-Clear Lake*

**Gary Smith**  
*Texas A & M University*

April 1992



*Research Institute for Computing and Information Systems  
University of Houston-Clear Lake*

**F • I • N • A • L   R • E • P • O • R • T**

## *The RICIS Concept*

---

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

# ***DATA MANAGEMENT SYSTEMS (DMS) COMPLEX DATA TYPES STUDY***

**APPENDICES  
TO  
FINAL REPORT**

**VOL 3 OF 3  
APPENDICES D (1-3) & E**



## **Appendix D-1**

### **Concurrent (Competing) Reader and Writer Performance Test**

**Full Protection of Semaphores and RODB Component.**



```

      t
      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t t
r      oooo  oooo  tt

```

```

      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr   r  e   e      a  d  dd
r      eeeee  aaaaa  d  d
r      e      a   a  d  d
r      e   e  a   aa  d  dd
r      eeee  aaaa a  ddd d

      m m mm  eeee
mm m m  e   e
m m m  eeeee
m m m  e
.. m m m  e   e
.. m m m  eeee

```

Job: read.me  
Date: Sat Apr 11 23:09:58 1992

THIS IS A TEST OF CONCURRENT (COMPETING) READERS AND WRITERS.  
THERE ARE THREE TASKS RUNNING IN THIS SYSTEM: TWO READERS AND ONE WRITER.  
THEY ARE ACCESSING THE RODB COMPONENT WHICH IS PROTECTED BY A MECHANISM.  
IN RODBSTD1, ALL THE TASKS HAVE THE SAME PRIORITIES. THE RESULTS ARE IN  
FILES RODBCOMP1.OUT (RODBCOMP11.OUT AND RODBCOMP12.OUT). THESE RESULTS  
CORRESPOND TO THE INPUT FILES RODBCOMP1.IN (RODBCOMP11.IN AND RODBCOMP12.IN)

This directory stores all the files to build up RODB "attribute" components.  
The protection mechanism is that locking is set at the RODB level. During the  
lock setting, there is prevention of preemption. This is done inside the  
C function by using the fast\_setprio system call. There is only one set of  
three UNIX semaphores in the whole system. Before actual reading, a set of five  
semaphore operations are imposed on the semaphores. After a read completes one  
semaphore operation is imposed on a semaphore. Before actually writing,  
there are two levels of semaphore operations: write-desire and write-lock.  
For write-desire one semaphore operation is imposed (test&set) on a semaphore.  
Once the read-lock semaphore is unlocked, (i.e. last reader exits), then a  
writer can enter and lock out all new readers and any other following writer.  
Write-lock imposes a set of four semaphore operations on the semaphores.  
After actual writing, a set of two semaphore operations are imposed on the  
semaphores (i.e. unlock for readers or another writer). This system gives  
preferences to writers but readers actually reading lock out any writers  
that are waiting.



```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr   r    o   o    o   o
r        o   o    o   o
r        o   o    o   o
r        o   o    o   o
r        oooo      oooo

```

```

      d   b
      d   b
      d   b
      ddd d b bbb
      d dd bb   b
      d   d b     b
      d   d b     b
      d dd bb   b
      ddd d b bbb
      tt

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      ssss
      s   s
      ss
      ss
      s   s
      ssss

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      DDDDD
      D   D
      D   D
      D   D
      D   D
      D   D
      D   D
      DDDDD

      1
      11
      1 1
      1
      1
      1
      1
      1
      1111

```

Job: rodbtstD1.ada  
 Date: Sat Apr 11 22:20:04 1992

```

-- This is the concurrent reading and writing test program with default priori
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
with RODB_Test_Data1;
procedure RodbtstD1 is

```

```

-- Constant definitions

```

```

ATTR_SIZE          : constant integer := 200;
NUMBER_OF_TIMES1   : constant integer := RODB_Test_Data1.Number_Of_Times1;
NUMBER_OF_TIMES2   : constant integer := RODB_Test_Data1.Number_Of_Times2;
NUMBER_OF_TIMES3   : constant integer := RODB_Test_Data1.Number_Of_Times3;

```

```

-- Package instantiation

```

```

package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package FIX_IO is new TEXT_IO.FIXED_IO(duration);
package RCDT renames RODB_Component_Data_Types;
package ROBCP renames RODB_Component;

```

```

-- task declaration

```

```

task Reader1 is
    entry Finish;
end Reader1;
task Reader2 is
    entry Finish;
end Reader2;
task Writer is
    entry Finish;
end Writer;

```

```

-- Variable definition

```

```

Start_Time1 : CALENDAR.time;
Start_Time2 : CALENDAR.time;
Start_Time3 : CALENDAR.time;
Finish_Time1 : CALENDAR.time;
Finish_Time2 : CALENDAR.time;
Finish_Time3 : CALENDAR.time;
Result1      : duration;
Result2      : duration;
Result3      : duration;
Addr_List1   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
Addr_List2   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
Addr_List3   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
Attr_List1   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200),others=>(0,200));
Attr_List2   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200),others=>(0,200));
Attr_List3   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200),others=>(0,200));
Length1      : integer := 1;
Length2      : integer := 1;
Length3      : integer := 1;
Outfile      : file_type;

```

```

-- The body of task reader1

```

```

task body Reader1 is
begin
    Start_Time1 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES1 loop
        ROBCP.Read_Attrs(Addr_List1, Length1, Attr_List1);
    end loop;
    Finish_Time1 := CALENDAR.clock;
    Result1      := Finish_Time1 - Start_Time1;

```

```

    accept Finish;
exception
    when others =>
        put_line("Task Reader1 has an exception.");
end Reader1;

-- The body of task reader2
task body Reader2 is
begin
    Start_Time2 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES2 loop
        RODBCP.Read_Attrs(Addr_List2, Length2, Attr_List2);
    end loop;
    Finish_Time2 := CALENDAR.clock;
    Result2      := Finish_Time2 - Start_Time2;

    accept Finish;
exception
    when others =>
        put_line("Task Reader2 has an exception.");
end Reader2;

-- The body of task writer
task body Writer is
begin
    Start_Time3 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES3 loop
        RODBCP.Write_Attrs(Addr_List3, Length3, Attr_List3);
    end loop;
    Finish_Time3 := CALENDAR.clock;
    Result3      := Finish_Time3 - Start_Time3;

    accept Finish;
exception
    when others =>
        put_line("Task Writer has an exception.");
end Writer;

begin

    -- Terminate gracefully
    Reader1.Finish;
    Reader2.Finish;
    Writer.Finish;

    -- Write out the results
    create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "Task      Number_Of_Iterations      Times");
    put(Outfile, "Reader1  ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES1);
    FIX_IO.put(Outfile, Result1);
    new_line(Outfile);
    put(Outfile, "Reader2  ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES2);
    FIX_IO.put(Outfile, Result2);
    new_line(Outfile);
    put(Outfile, "Writer   ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES3);
    FIX_IO.put(Outfile, Result3);

```

```
new_line(Outfile);  
close(Outfile);  
end RodbtstD1;
```

[illegible]

```
Job:  rodbcomp.dat
Date:  Sat Apr 11 22:27:31 1992
```

```
10
0 100
1 200
2 300
3 400
4 500
5 600
6 700
7 800
8 900
9 1000
10
0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
8 I
9 J
10
0 false
1 false
2 false
3 false
4 false
5 false
6 false
7 false
8 false
9 false
10
0 100.0
1 200.0
2 300.0
3 400.0
4 500.0
5 600.0
6 700.0
7 800.0
8 900.0
9 1000.0
```

cccc      oooo      m m mm      p ppp

c      c      o      o      mm m m      pp p

c      c      o      o      m m m      p p

c      c      o      o      m m m      p p

c      c      o      o      m m m      pp p

cccc      oooo      m m m      p ppp

p

p

p

---

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE   : constant integer := 10;
  CHAR_SIZE  : constant integer := 10;
  BOOL_SIZE  : constant integer := 10;
  FLT_SIZE   : constant integer := 10;
  SHMKEY     : constant integer := 99;
  SEMKEY     : constant integer := 100;
  SHM_SIZE   : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET : constant integer := INT_SIZE*4;
  BOOL_OFFSET : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET  : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY : in integer;
                 SIZE : in integer;
                 FLAG : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID : in integer;
                SHMADDR : in system.address;
                FLAG : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```



```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

```

-- Semaphore system call and C function interface

```

function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

```

```

end RODB_Component_Data_Types;

```

```

                d   b
                d   b
                d   b
r rrr      oooo      ddd d   b bbb
rr   r      o   o      d   dd  bb   b
r         o   o      d   d   b     b
r         o   o      d   d   b     b
r         o   o      d   dd  bb   b
r         oooo      ddd d   b bbb

```

```

cccc      oooo      m m mm      p ppp
c        c      o      o      mm m m      pp      p
c        o      o      m m m      p      p
c        o      o      m m m      p      p
c        o      o      m m m      pp      p
      cccc      oooo      m m m      p ppp
      p
      p
      p

```

Job: rodb\_component .ada  
Date: Sat Apr 11 22:29:11 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in      RCDT.Attr_List_Type);

    -- Print out the semaphore values
    procedure Print_Sems;

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end Rodb_Component;

```

```

                d      b
                d      b
                d      b
r rrr          oooo    ddd d      b bbb
rr   r        o   o    d   dd     bb   b
r          o   o    d   d      b       b
r          o   o    d   d      b       b
r          o   o    d   dd     bb   b
r          oooo    ddd d      b bbb

```

```

cccc      oooo      m m mm      p ppp
c        c      o    o      mm m   m      pp   p
c        c      o    o      m   m   m      p    p
c        c      o    o      m   m   m      p    p
c        c      o    o      m   m   m      pp   p
cccc      oooo      m   m   m      p ppp
                p
                p
                p

```

Job: rodb\_component.ada  
Date: Sat Apr 11 22:29:31 1992

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READBEG(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READEND(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

Temp : system.address;
Flag : integer;
begin
-- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
Flag := RCDT.WRITEBEG(Semid);
-- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
if Flag = -1 then
raise Sem_Exception;
end if;
for I in 1.. Length loop
if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
raise Shm_Outrange;
end if;
Temp := Shmaddr + system.offset(Addr_List(I));
if (Addr_List(I) < RCDT.CHAR_OFFSET) then
RCDT.AINT(Temp, Attr_List(I).Int_Value);
elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
else
RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
end if;
end loop;
-- delay 10.0;
-- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
Flag := RCDT.WRITEEND(Semid);
-- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
if Flag = -1 then
raise Sem_Exception;
end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
Flag : integer;
begin
Flag := RCDT.SEMPRINT(Semid);
if Flag = -1 then
raise Sem_Exception;
end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--      Number_Of_Integers
--      Position1 Integer1
--      Position2 Integer2
--      ...
--      Number_Of_Characters
--      Position1 Character1
--      Position2 Character2
--      ...
--      Number_Of_Booleans
--      Position1 Boolean1
--      Position2 Boolean2
--      ...
--      Number_Of_Floats
--      Position1 Float1
--      Position2 Float2

```

```

--      ...
procedure Load_Comps(Filename : in      string) is
  Infile : FILE_TYPE;
  Temp   : system.address;
  Flag   : integer;
begin
  open(Infile, in_file, Filename);

  -- Initialize RODB Integer Component
  for I in 1..RCDT.INT_SIZE loop
    Temp := Shmaddr + system.offset((I-1)*4);
    RCDT.AINT(Temp, 0);
  end loop;
  Load_Ints(Infile);

  -- Initialize RODB Character Component
  for I in 1..RCDT.CHAR_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
  end loop;
  Load_Chars(Infile);

  -- Initialize RODB Boolean Component
  for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
  end loop;
  Load_Bools(Infile);

  -- Initialize RODB Float Component
  for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
  end loop;
  Load_Flts(Infile);

  close(Infile);

  Flag := RCDT.SEMSINIT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;

exception
  when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
  when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
  when Sem_Exception =>
    put_line("Semaphore cannot be initialized.");
    raise Sem_Exception;
  when others =>
    put_line("Unknown exception.");
    put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
           form=>"world=>read, owner=>read_write");
    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint_error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```



pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
    Length      : Integer;
    Temp_Pos     : integer;
    Temp_Int     : integer;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        INT_IO.get(Infile, Temp_Int);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
        RCDT.AINT(Temp_Addr, Temp_Int);
    end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
    Length      : Integer;
    Temp_Pos     : integer;
    Temp_Char    : character;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        get(Infile, Temp_Char);           -- Skip a space
        get(Infile, Temp_Char);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
        RCDT.ACHAR(Temp_Addr, Temp_Char);
    end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
    Length      : Integer;
    Temp_Pos     : integer;
    Temp_Bool    : boolean;
    Temp_Addr    : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        BOOL_IO.get(Infile, Temp_Bool);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

```

-- Load all the floats from a disk file to RODB Float Component

```

procedure Load_Flts(Infile : in FILE_TYPE) is
    Length : Integer;
    Temp_Pos : integer;
    Temp_Flt : float;
    Temp_Addr : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

```

pragma page;

-- Save all the integers from RODB Integer Component to a disk file

```

procedure Save_Ints(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

-- Save all the characters from RODB Character Component to a disk file

```

procedure Save_Chars(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```

```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;

    -- Initialize the RODB Components
    Load_Comps("rodbcomp.dat");

end Rodb_Component;

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
      oooo
      oooo
      oooo
      oooo

```

```

      d
      d
      d
      ddd d
      d dd
      d d
      d d
      d dd
      ddd d
      b
      b
      b
      b bbb
      bb b
      b b
      bb b
      b bbb
      eeee
      e e
      eeee
      e e
      eeee
      ggg g
      gg
      g
      g
      gg
      ggg g
      g
      g
      gggg

cccc
c c
c
c
c c
cccc

..
..

```

Job: readbeg.c  
Date: Sat Apr 11 23:40:14 1992

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */ /* Guarantee atomic ops */
    my_tid = gettid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform three semaphore operations */ /* Two ops are repeated */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[4] = RREAD_START; /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 5); /* Lock the critical section */
    if (flag == -1) {
        perror("readbeg fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

t
t
ttttt
t
t
t
t  t
  tt

```

```

      d
      d
      d
ddd d
d   dd
d   d
d   d
d   dd
ddd d

```

• •

• •

```

/* File: readend.c This is read end subroutine to reset protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t  my_tid;
    int     my_prio;

    /* Raise the priority to prevent preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);      /* Unlock critical section */
    if (flag == -1) {
        perror("readend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr  r      o      o      o      o
r          o      o      o      o
r          o      o      o      o
r          o      o      o      o
r          oooo      oooo

```

```

      i          t          b
      t          b
      t          b
      ttttt      eeee      b bbb      eeee      ggg g
      t          e      e      bb  b      e      e      g  gg
      t          eeeee      b      b      eeeee      g  g
      t          e      e      b      b      e      e      g  g
      t t        e      e      bb  b      e      e      g  gg
      tt         eeee      b bbb      eeee      ggg g
                                     g  g
                                     gggg

w      w      r rrr      ii
w w w      rr  r      i
w w w      r          i
w w w      r          i
w w w      r          i
ww ww      r          iii

      t          t          b
      t          b
      t          b
      t t        eeee      b bbb      eeee      ggg g
      tt         eeee      b bbb      eeee      ggg g
                                     g  g
                                     gggg

```

Job: writebeg.c  
Date: Sat Apr 11 23:40:11 1992



```

/* File: writebeg.c This is the write begin subroutine to set protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0}; /*lock out another writer*/
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE     = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0}; /* unlock write desire# */
/* #Guarantees writer progress */

int writebeg(semid)
int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int flag;
    void perror();
    tid_t my_tid;
    int my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK; /* preventing succeeding readers */
    sole_writer[2] = WREAD_START; /* preventing succeeding writers */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request# */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }
    /* #Note cancelling the write-request allows another writer to lock it */

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t t
r          oooo      oooo      tt

```

```

                                d
                                d
                                d
                                d
w      w  r rrr      ii      ttttt      eeee      eeee      n nnn      ddd d
w w w  rr  r      i      t      e   e      e   e      nn  n      d  dd
w w w  r          i      t      eeeee      eeeee      n   n      d  d
w w w  r          i      t      e       e      e       e      n   n      d  d
w w w  r          i      t t      e   e      e   e      n   n      d  dd
  ww ww  r      iii      tt      eeee      eeee      n   n      ddd d      ..

```

Job: writeend.c  
 Date: Sat Apr 11 23:40:13 1992

```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    fast_setprio(my_tid, 31);

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }

    /* Lower the priority to the normal */
    fast_setprio(my_tid, my_prio);

    return flag;
}

```

[illegible]

Job: rodbcomp.dat  
Date: Sat Apr 11 22:34:04 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0

[illegible]

Job: rodbcomp11.in  
Date: Sat Apr 11 23:02:10 1992

2500 2500 5000

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b  bbb  cccc  oooo  m m mm  p ppp  1
rr  r  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p  11
r      o  o  d  d  b  b  c  c  o  o  m m m m  p  p  1
r      o  o  d  d  b  b  c  c  o  o  m m m m  p  p  1
r      oooo  ddd d  b  bbb  cccc  oooo  m m m  p ppp  1
      11111
      p
      p
      p

```

Job: rodbcomp11.out  
Date: Sat Apr 11 23:02:22 1992



Task	Number_Of_Iterations	Times
Reader1	2500	12.98090
Reader2	2500	12.98090
Writer	5000	17.08582

```

r rrr      oooo      ddd d      d      b      cccc      oooo      m m mm      p ppp      1
rr   r      o   o      d   dd      d      b      c   c      o   o      mm m m      pp   p      11
r          o   o      d   d      d      b      c       c      o   o      m   m   m      p     p      1 1
r          o   o      d   d      d      b      c       c      o   o      m   m   m      p     p      1
r          o   o      d   dd      d      bb   b      c       c      o   o      m   m   m      pp    p      1
r          oooo      ddd d      b   bbb      cccc      oooo      m   m   m      p   ppp      11111
          p
          p
          p

```

Job: rodbcomp12.in  
Date: Sat Apr 11 23:02:41 1992

5000 5000 5000

[illegible]

Job: rodbcomp12.out  
Date: Sat Apr 11 23:03:17 1992

Task	Number_Of_Iterations	Times
Reader1	5000	25.99048
Reader2	5000	25.98096
Writer	5000	25.99048



## **Appendix D-2**

### **Concurrent (Competing)**

#### **Reader and Writer Performance Test**

**No Prevention of Preemption but with Semaphore Protection of RODB Component.**





```

      t
      t
r rrr  oooo  oooo  ttttt
rr   r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t  t
r      oooo  oooo  tt

```

```

      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr   r  e   e  a   a  d   dd
r      eeeee  aaaaa  d   d
r      e      a   a  d   d
r      e   e  a   aa  d   dd
r      eeee  aaaa a  ddd d

```

```

m m mm  eeee
mm m m  e   e
m m m  eeeee
m m m  e
.. m m m  e   e
.. m m m  eeee

```

Job: read.me  
Date: Mon Apr 20 17:47:34 1992

THERE ARE THREE TASKS RUNNING IN THE SYSTEM: TWO READERS AND ONE WRITER. THEY ARE ACCESSING THE RODB COMPONENT WHICH IS PROTECTED BY A MECHANISM. IN RODBTST1, ALL THE TASKS HAVE THE SAME PRIORITIES. THE RESULTS ARE IN FILES RODBCOMP1.OUT (RODBCOMP11.OUT AND RODBCOMP12.OUT) for 5000 reads and 5000 writes as well as 10000 reads and 5000 writes respectively. THESE FILES CORRESPOND to INPUT FILES RODBCOMP1.IN (RODBCOMP11.IN and RODBCOMP12.IN).

This directory stores all the files to build up RODB "attribute" components. The protection mechanism is that locking is set at the RODB level. During the lock setting, there is NO prevention of preemption. Inside the C functions the fast\_setprio system calls are commented out. There is only one set of three UNIX semaphores in the whole system. Before actual reading, a set of five semaphore operations are imposed on the semaphores. After a read, one semaphore operation is imposed on a semaphore (decreasing reader count). Before writing there are two levels of semaphore operations which are imposed; namely, write-desire and write-lock. For write-desire only one semaphore operation is imposed on its semaphore (test&lock) and when that is set and the last reader has finished, then, write-lock is set as one of a set of four semaphore operations imposed on the semaphores. After the writer finishes writing, a set of two semaphore operations are imposed on the semaphores (unlocking the RODB to allow readers or other writer in). THE INTENT IS TO DETERMINE THE EFFECT OF NOT USING PRIORITY RAISE/LOWER MANIPULATION WHICH FORMERLY GUARANTEED THE ATOMIC NATURE OF THE SEMAPHORE OPERATIONS. THIS TEST IS WITHOUT THAT COST.

r rrr	oooo	oooo	t t t t t
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

		d	b
		d	b
		d	b
r rrr	oooo	ddd d	b bbb
rr r	o o	d dd	bb b
r	o o	d d	b b
r	o o	d d	b b
r	o o	d dd	bb b
r	oooo	ddd d	b bbb

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp p
c	o o	m m m	p p
c	o o	m m m	p p
c c	o o	m m m	pp p
cccc	oooo	m m m	p ppp
			p
			p
			p

Job: rodb\_component\_data\_types\_.ada  
Date: Mon Apr 13 20:37:49 1992

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY : in integer;
                  SIZE : in integer;
                  FLAG : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID : in integer;
                 SHMADDR : in system.address;
                 FLAG : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD    : in integer;
                BUFF   : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

-- Semaphore system call and C function interface
function SEMGET(KEY    : in integer;
                NSEMS  : in integer;
                FLAG    : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

end RODB_Component_Data_Types;

```

```

                                     d  b
                                     d  b
                                     d  b
r rrr      oooo      ddd d  b bbb
rr  r      o    o    d  dd  bb  b
r          o    o    d    d  b    b
r          o    o    d    d  b    b
r          o    o    d  dd  bb  b
r          oooo      ddd d  b bbb

```

[illegible]

Job: rodb\_component .ada  
Date: Mon Apr 13 20:37:51 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

    -- Package renaming
    package RCDT renames Rodb_Component_Data_Types;

    -- Exception definition
    Shm_Exception : exception;
    Shm_Outrange  : exception;
    Sem_Exception  : exception;

    -- Read attributes from RODB components
    procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in out RCDT.Attr_List_Type);

    -- Write attributes to RODB components
    procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length      : in      integer;
                        Attr_List   : in      RCDT.Attr_List_Type);

    -- Print out the semaphore values
    procedure Print_Sems;

    -- Load RODB components from a disk file
    procedure Load_Comps(Filename : in      string);

    -- Save RODB components to a disk file
    procedure Save_Comps(Filename : in      string);

    -- Shutdown the RODB components
    procedure Shutdown_Comps;

end RODB_COMPONENT;

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
      oooo
      o
      o
      o
      o
      oooo
      o
      o
      o
      o
      oooo

```

```

      d
      d
      d
      d
      b
      b
      b
      b bbb
      bb  b
      b    b
      b    b
      bb  b
      b bbb

r rrr
rr  r
r
r
r
r
      oooo
      o
      o
      o
      o
      oooo
      ddd d
      d dd
      d  d
      d  d
      d dd
      ddd d
      ddd d
      b bbb

```

```

      cccc
      c  c
      c
      c
      c  c
      cccc

      oooo
      o  o
      o  o
      o  o
      o  o
      oooo

      m m mm
      mm m m
      m m m
      m m m
      m m m
      m m m

p ppp
pp  p
p    p
p    p
pp  p
p ppp
p
p
p
p

```

Job: rodb\_component.ada  
Date: Mon Apr 13 20:37:53 1992



```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is

```

```

    Temp : system.address;
    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READBEG(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.READEND(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

Temp : system.address;
Flag : integer;
begin
  -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
  Flag := RCDT.WRITEBEG(Semid);
  -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
  if Flag = -1 then
    raise Sem_Exception;
  end if;
  for I in 1.. Length loop
    if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp := Shmaddr + system.offset(Addr_List(I));
    if (Addr_List(I) < RCDT.CHAR_OFFSET) then
      RCDT.AINT(Temp, Attr_List(I).Int_Value);
    elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
      RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
    elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
      RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
    else
      RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
    end if;
  end loop;
  -- delay 10.0;
  -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
  Flag := RCDT.WRITEEND(Semid);
  -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
  Flag : integer;
begin
  Flag := RCDT.SEMPRINT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2

```

```

--      ...
procedure Load Comps(Filename : in      string) is
  Infile : FILE_TYPE;
  Temp   : system.address;
  Flag   : integer;
begin

  open(Infile, in_file, Filename);

  -- Initialize RODB Integer Component
  for I in 1..RCDT.INT_SIZE loop
    Temp := Shmaddr + system.offset((I-1)*4);
    RCDT.AINT(Temp, 0);
  end loop;
  Load_Ints(Infile);

  -- Initialize RODB Character Component
  for I in 1..RCDT.CHAR_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
  end loop;
  Load_Chars(Infile);

  -- Initialize RODB Boolean Component
  for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
  end loop;
  Load_Bools(Infile);

  -- Initialize RODB Float Component
  for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
  end loop;
  Load_Flts(Infile);

  close(Infile);

  Flag := RCDT.SEMSINIT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;

exception
  when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
  when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
  when Sem_Exception =>
    put_line("Semaphore cannot be initialized.");
    raise Sem_Exception;
  when others =>
    put_line("Unknown exception.");
    put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
           form=>"world=>read, owner=>read_write");
    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint_error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```

pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Int    : integer;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
    RCDT.AINT(Temp_Addr, Temp_Int);
  end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Char   : character;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    get(Infile, Temp_Char);           -- Skip a space
    get(Infile, Temp_Char);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
    RCDT.ACHAR(Temp_Addr, Temp_Char);
  end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
Length      : Integer;
Temp_Pos    : integer;
Temp_Bool   : boolean;
Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    BOOL_IO.get(Infile, Temp_Bool);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in    FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

pragma page;

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```

```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;

    -- Initialize the RODB Components
    Load_Comps("rodbcomp.dat");
end Rodb_Component;

```

```

                                     d  b
                                     d  b
                                     d  b
r rrr      oooo      ddd d  b bbb
rr      r  o      o  d  dd  bb  b
r      o      o  d  d  b      b
r      o      o  d  d  b      b
r      o      o  d  dd  bb  b
r      oooo      ddd d  b bbb

```

```

      t      t      t      t
      t      t      t      t
  ttttt      eeee      ssss      ttttt
      t      e      s      s      t
      t      eeeee     ss      t
      t      e      ss      t
      t  t      e      s      s      t  t
        tt      eeee      ssss      tt

```

Job: rodb\_test\_data1.ada  
Date: Mon Apr 13 20:38:44 1992



```

with TEXT_IO;
use TEXT_IO;
package RODB_Test_Data1 is

    Number_Of_Times1 : integer;
    Number_Of_Times2 : integer;
    Number_Of_Times3 : integer;

    package INT_IO is new TEXT_IO.INTEGER_IO(integer);

end RODB_Test_Data1;

with TEXT_IO;
use TEXT_IO;
package body RODB_Test_Data1 is
    Infile : file_type;
begin
    open(Infile, in_file, "rodbcompl.in");
    INT_IO.get(Infile, Number_Of_Times1);
    INT_IO.get(Infile, Number_Of_Times2);
    INT_IO.get(Infile, Number_Of_Times3);
    close(Infile);
end RODB_Test_Data1;

```

```

r rrr      oooo      ddd d  b bbb      ttttt      ssss      ttttt      EEEEEEE      1
rr   r      o   o      d  dd  bb   b      t      s   s      t      E      11
r          o   o      d    d  b     b      t      ss      t      E      1 1
r          o   o      d    d  b     b      t      ss      t      E      1
r          o   o      d  dd  bb   b      t  t  s   s      t  t  E      1
r          oooo      ddd d  b bbb      tt      ssss      tt      EEEEEEE      11111

```

Job: rodbtstE1.ada  
Date: Mon Apr 13 20:49:27 1992

```

-- This is the concurrent reading and writing test program
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
with RODB_Test_Data1;
procedure RodbTest1 is

    -- Constant definitions
    ATTR_SIZE      : constant integer := 200;
    NUMBER_OF_TIMES1 : constant integer := RODB_Test_Data1.Number_Of_Times1;
    NUMBER_OF_TIMES2 : constant integer := RODB_Test_Data1.Number_Of_Times2;
    NUMBER_OF_TIMES3 : constant integer := RODB_Test_Data1.Number_Of_Times3;

    -- Package instantiation
    package INT_IO is new TEXT_IO.INTEGER_IO(integer);
    package FIX_IO is new TEXT_IO.FIXED_IO(duration);
    package RCDT renames RODB_Component_Data_Types;
    package ROBCP renames RODB_Component;

    -- task declaration
    task Reader1 is
        entry Finish;
    end Reader1;
    task Reader2 is
        entry Finish;
    end Reader2;
    task Writer is
        entry Finish;
    end Writer;

    -- Variable definition
    Start_Time1 : CALENDAR.time;
    Start_Time2 : CALENDAR.time;
    Start_Time3 : CALENDAR.time;
    Finish_Time1 : CALENDAR.time;
    Finish_Time2 : CALENDAR.time;
    Finish_Time3 : CALENDAR.time;
    Result1      : duration;
    Result2      : duration;
    Result3      : duration;
    Addr_List1   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
    Addr_List2   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
    Addr_List3   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
    Attr_List1   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
    Attr_List2   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
    Attr_List3   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
    Length1      : integer := 1;
    Length2      : integer := 1;
    Length3      : integer := 1;
    Outfile      : file_type;

    -- The body of task reader1
    task body Reader1 is
    begin
        Start_Time1 := CALENDAR.clock;
        for I in 1..NUMBER_OF_TIMES1 loop -- Read is without preemption protection
            ROBCP.Read_Attrs(Addr_List1, Length1, Attr_List1);
        end loop;
        Finish_Time1 := CALENDAR.clock;
        Result1      := Finish_Time1 - Start_Time1;

```

```

    accept Finish;
exception
    when others =>
        put_line("Task Reader1 has an exception.");
end Reader1;

-- The body of task reader2
task body Reader2 is
begin
    Start_Time2 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES2 loop
        RODBCP.Read_Attrs(Addr_List2, Length2, Attr_List2);
    end loop;
    Finish_Time2 := CALENDAR.clock;
    Result2      := Finish_Time2 - Start_Time2;

    accept Finish;
exception
    when others =>
        put_line("Task Reader2 has an exception.");
end Reader2;

-- The body of task writer
task body Writer is
begin
    Start_Time3 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES3 loop -- Similarly no preemption protection
        RODBCP.Write_Attrs(Addr_List3, Length3, Attr_List3);
    end loop;
    Finish_Time3 := CALENDAR.clock;
    Result3      := Finish_Time3 - Start_Time3;

    accept Finish;
exception
    when others =>
        put_line("Task Writer has an exception.");
end Writer;

begin

    -- Terminate gracefully
    Reader1.Finish;
    Reader2.Finish;
    Writer.Finish;

    -- Write out the results
    create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "Task      Number_Of_Iterations      Times");
    put(Outfile, "Reader1  ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES1);
    FIX_IO.put(Outfile, Result1);
    new_line(Outfile);
    put(Outfile, "Reader2  ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES2);
    FIX_IO.put(Outfile, Result2);
    new_line(Outfile);
    put(Outfile, "Writer   ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES3);
    FIX_IO.put(Outfile, Result3);

```

```
new_line(Outfile);  
close(Outfile);  
  
end RodbtstE1;
```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          o  o      o  o      tt
r          oooo      oooo

```

```

      d  b
      d  b
      d  b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr  r      o  o      d  dd      bb  b      c  c      o  o      mm m  m      pp  p
r          o  o      d  d      b  b      c  c      o  o      m  m  m      p  p
r          o  o      d  dd      bb  b      c  c      o  o      m  m  m      pp  p
r          oooo      ddd d      b bbb      cccc      oooo      m  m  m      p ppp
r          oooo      ddd d      b bbb      cccc      oooo      m  m  m      p ppp

```

..

..

p

p

p

Job: rodbcomp.dat  
Date: Mon Apr 13 20:49:49 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

			d	b						1
			d	b						11
			d	b						1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp			1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p			1
r	o o	d d	b b	c	o o	m m m	p p			1
r	o o	d d	b b	c	o o	m m m	p p			1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	pp p			1
							p ppp			11111
							p			
							p			
							p			

Job: rodbcomp11.in  
Date: Mon Apr 13 20:48:48 1992



2500 2500 5000

```

      1
     11
    1 1
      1
      1
      1
      1
      1
    11111

```

[illegible]

Job: rodbcomp11.out  
Date: Mon Apr 13 20:49:01 1992

Task	Number_Of_Iterations	Times
Reader1	2500	5.66620
Reader2	2500	5.58038
Writer	5000	6.98090

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          oooo      oooo      tt

```

```

                        d  b
                        d  b
                        d  b
r rrr      oooo      ddd d  b bbb      cccc      oooo      m m mm      p ppp      1
rr  r      o  o      d  dd  bb  b      c  c      o  o      mm m m      pp  p      11
r          o  o      d  d  b  b      c      o  o      m m m      p  p      1
r          o  o      d  dd  bb  b      c      o  o      m m m      pp  p      1
r          oooo      ddd d  b bbb      cccc      oooo      m m m      p ppp      11111

```

Job: rodbcomp12.in  
 Date: Mon Apr 13 20:48:54 1992

5000 5000 5000

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb  cccc  oooo  m m mm  p ppp
rr  r  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p
r      o  o  d  d  b  b  b  c  c  o  o  m m m  p  p
r      o  o  d  d  b  b  b  c  c  o  o  m m m  p  p
r      o  o  d  dd  bb  b  c  c  o  o  m m m  pp  p
r      oooo  ddd d  b bbb  cccc  oooo  m m m  p ppp
                                     p
                                     p
                                     p

```

1  
11  
1 1  
1  
1  
1  
1  
11111

Job: rodbcomp12.out  
Date: Mon Apr 13 20:49:10 1992

Task	Number_Of_Iterations	Times
Reader1	5000	11.41821
Reader2	5000	11.19073
Writer	5000	7.17981

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          o  o      o  o      tt
r          oooo      oooo

```

```

                        d  b
                        d  b
                        d  b
r rrr      eeee      aaaa      ddd d      b bbb      eeee      ggg g      cccc
rr  r      e   e      a      d  dd      bb  b      e   e      g  gg      c   c
r          eeeee      aaaaa      d  d      b   b      eeeee      g  g      c
r          e   e      a   a      d  dd      bb  b      e   e      g  gg      c
r          eeee      aaaa a      ddd d      b bbb      eeee      ggg g      c
r          eeee      aaaa a      ddd d      b bbb      eeee      ggg g      c
                        g  g
                        gggg

```

Job: readbeg.c  
 Date: Mon Apr 13 20:49:59 1992



```

/* File: readbeg.c This is read begin subroutine to set reading protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int    flag;
    void    perror();
    tid_t    my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */ /* Not this time! */
    /* the fast setting of the priority is commented out */

    /* Perform three semaphore operations */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[4] = RREAD_START; /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 5); /* Lock the critical section */
    if (flag == -1) {
        perror("readbeg fails: ");
    }

    /* Lower the priority to the normal */
    /* fast_setprio(my_tid, my_prio); */
    /* the lowering of the priority is also commented out */
    return flag;
}

```

```

      d
      d
      d
      d
r rrr      eeee      aaaa      ddd d      eeee      n nnn      ddd d
rr  r      e      e      a      d      dd      e      e      nn      n      d      dd      cccc
r      eeeeee      aaaaaa      d      d      eeeeee      n      n      d      d      c
r      e      a      a      d      d      e      n      n      d      d      c
r      e      e      a      aa      d      dd      e      e      n      n      d      dd      ..      c
r      eeee      aaaa a      ddd d      eeee      n      n      ddd d      ..      cccc

```

```
Job: readend.c
Date: Mon Apr 13 20:50:16 1992
```

```

/* File: readend.c This is read end subroutine to release protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */ /* Not here !! */
    my_tid = getstid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */
    /* The fast setting of the priority is commented out */

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);      /* Unlock critical section */
    if (flag == -1) {
        perror("readend fails: ");
    }

    /* Lower the priority to the normal */ /* Also not here either !! */
    /* fast_setprio(my_tid, my_prio); */
    /* The lowering of the priority is commented out here also */
    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r    o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t
r        o   o    o   o    t t
r        oooo     oooo     tt

```

```

              i          t          b
              t          t          b
              t          t          b
w      w  r rrr      ii      ttttt      eeee      b bbb      eeee      ggg g
w  w  w  rr   r      i          t          e   e      bb   b      e   e      g   gg
w  w  w  r        i          t          eeeee     b   b      eeeee     g   g
w  w  w  r        i          t          e          b   b      e          g   gg
w  w  w  r        i          t t         e   e      bb   b      e   e      g   gg
  ww ww  r      iii          tt         eeee      b bbb      eeee     ggg g
                                     gggg
                                     g
                                     gggg

```

Job: writebeg.c  
Date: Mon Apr 13 20:50:26 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0};
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE     = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */ /* Disabled here !! */
    my_tid = getstid();
    my_prio = getprio(getpid());
    /* fast setprio(my_tid, 31); */
    /* The fast setting of the priority is commented out here */

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding readers */
    sole_writer[2] = WREAD_START;      /* preventing succeeding writers */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }

    /* Lower the priority to the normal */ /* Not necessary since not raised */
    /* fast setprio(my_tid, my_prio); */
    /* The lowering of the priority is commented out */
    return flag;
}

```

```

                                d
                                d
                                d
                                d
w      w  r rrr      ii      ttttt      eeee      eeee      n nnn      ddd d
w  w  w  rr      r      i      t      e      e      nn      n      d      dd
w  w  w  r      i      t      e      e      n      n      d      d
w  w  w  r      i      t      e      e      n      n      d      d
w  w  w  r      i      t  t      e      e      n      n      d      dd      ..
  ww ww  r      iii      tt      eeee      eeee      n      n      ddd d      ..

```

```
Job:  writeend.c
Date:  Mon Apr 13 20:50:52 1992
```

```

/* File: writeend.c This is write end subroutine to reset write protection */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t  my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */ /* Not here !! */
    my_tid = getstid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */
    /* The fast setting of the priority is commented out */

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;          /* Allow writer in */
    sole_writer[1] = WWRITE_UNLOCK;      /* Allow reader in */
    flag = semop(semid, sole_writer, 2); /* Unlock critical section */
    if (flag == -1) {
        perror("writeend fails: ");
    }

    /* Lower the priority to the normal */ /* Not necessary since not raised */
    /* fast_setprio(my_tid, my_prio); */
    /* The lowering of the priority is commented out */
    return flag;
}

```

			t
			t
r rrr	oooo	oooo	ttttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

				i		i	t	
							t	
ssss	eeee	m m mm	ssss	ii	n nnn	ii	ttttt	
s s	e e	mm m m	s s	i	nn n	i	t	
ss	eeeeee	m m m	ss	i	n n	i	t	
ss	e	m m m	ss	i	n n	i	t	
s s	e e	m m m	s s	i	n n	i	t t	..
ssss	eeee	m m m	ssss	iii	n n	iii	tt	..

Job: semsinit.c  
Date: Mon Apr 13 20:51:09 1992



```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr  r      o      o      o      o
r          o      o      o      o
r          o      o      o      o
r          o      o      o      o
r          oooo      oooo

```

```

ssss      eeee      m m mm      ssss      r rrr      m m mm      v      v      cccc
s      s      e      e      mm m m      s      s      rr  r      mm m m      v      v      c      c
ss          eeeeeee      m m m      ss          r          m m m      v      v      c      c
ss          e          m m m      ss          r          m m m      v      v      c      c
s      s      e      e      m m m      s      s      r          m m m      v v      ..      c      c
ssss      eeee      m m m      ssss      r          m m m      v          ..      cccc

```

Job: semsrmv.c  
Date: Mon Apr 13 20:51:20 1992

```
/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```

```

      t
      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t
r          oooo      oooo      tt

```

```

                                     i
                                     t
                                     t
                                     t
s s s s      e e e e      m m m m      p p p p      r r r r      i i      n n n n      t t t t t
s   s      e   e      m m m m      p p   p      r r   r      i   n n   n      t
ss        e e e e e      m m m m      p   p      r          i   n   n      t
   ss      e   e      m m m m      p   p      r          i   n   n      t
s   s      e   e      m m m m      p p   p      r          i   n   n      t
s s s s      e e e e      m m m m      p p p p      r          i i i      n   n      t t
                                     p
                                     p
                                     p

```

Job: semprint.c  
 Date: Mon Apr 13 20:51:31 1992

```
/*File:semprint.c This is semaphore print subroutine to print semaphore values*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semprint(semid)
    int semid;
{
    short outarray[3];
    int    flag;
    void    perror();
    int    i;

    flag = semctl(semid, 3, GETALL, outarray);
    if (flag == -1) {
        perror("semprint fails: ");
    }
    for (i=0; i<3; ++i) {
        printf("Semaphore %d has the value of %d\n", i, outarray[i]);
    }
    return(flag);
}
```



## **Appendix D-3**

### **Simulated RODB**

#### **Concurrent (Competing), Reads and Writes**

**Prevention of Preemption Disabled**  
**Semaphore Protection Disabled**





```

      t
      t
      tttt
r rrr  oooo  oooo  t
rr  r  o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      o   o  o   o  t
r      oooo  oooo  tt

```

```

      d
      d
      d
r rrr  eeee  aaaa  ddd d
rr  r  e   e  a   d  dd
r      eeeee  aaaaa  d  d
r      e   a   a   d  d
r      e   e  a   aa  d  dd
r      eeee  aaaa a  ddd d

```

```

m m mm  eeee
mm m m  e   e
m m m  eeeee
m m m  e
m m m  e   e
m m m  eeee

```

Job: read.me  
Date: Wed Apr 15 19:28:49 1992

THERE ARE THREE TASKS RUNNING IN THE SYSTEM: TWO READERS AND ONE WRITER. THEY ARE ACCESSING TO THE RODB COMPONENT WHICH IS NOT NOW PROTECTED BY A MECHANISM. IN RodbtstF1 ALL THE TASKS HAVE THE SAME PRIORITY. THE RESULTS ARE IN FILES RODBCOMP1.OUT(RODBCOMP11.OUT AND RODBCOMP12.OUT). THE CORRESPONDING INPUT FILES ARE FILES RODBCOMP1.IN (RODBCOMP11.IN AND RODBCOMP12.IN).

This directory stores all the files to build up a RODB "attribute" components. The protection mechanism of locking at the RODB level is disabled. The prevention of preemption to protect the locking semaphores is also disabled. All of the protection was formerly done inside four C functions which used the fast\_setprio system call and the semop system call. There one set of three UNIX semaphores formerly used in the whole system. Now, all these system calls are disabled and no reader/writer protection is provided. While this would contribute to corrupt data items being read, the test was performed to see the time it would take for competing reads and writes in the "raw". Of course, now the overhead of calling the read and write beginning and ending functions is of little use, but they were left in the system to provide a method of isolating the costs of the protection mechanisms.

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr      oooo      oooo
rr      r  o      o  o      o
r          o      o  o      o
r          o      o  o      o
r          o      o  o      o
r          oooo      oooo

```

```

      d      b
      d      b
      d      b
      ddd d  b bbb
d      dd  bb  b
d      d  b      b
d      d  b      b
d      dd  bb  b
ddd d  b bbb

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      ssss
s      s
ss
ss
s      s
ssss

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

FFFFFFFFF      1
F          11
F          1 1
F          1
FFFFFFF      1
F          1
F          1
F          1
F          11111

```

Job: rodibtstF1.ada  
Date: Wed Apr 15 19:32:17 1992

```

-- This is the reading and writing test program
with TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
use TEXT_IO, CALENDAR, SYSTEM, RODB_Component_Data_Types, RODB_Component;
with RODB_Test_Data1;
procedure RodbtstF1 is

    -- Constant definitions
    ATTR_SIZE      : constant integer := 200;
    NUMBER_OF_TIMES1 : constant integer := RODB_Test_Data1.Number_Of_Times1;
    NUMBER_OF_TIMES2 : constant integer := RODB_Test_Data1.Number_Of_Times2;
    NUMBER_OF_TIMES3 : constant integer := RODB_Test_Data1.Number_Of_Times3;

    -- Package instantiation
    package INT_IO is new TEXT_IO.INTEGER_IO(integer);
    package FIX_IO is new TEXT_IO.FIXED_IO(duration);
    package RCDT renames RODB_Component_Data_Types;
    package RODBCP renames RODB_Component;

    -- task declaration
    task Reader1 is
        entry Finish;
    end Reader1;
    task Reader2 is
        entry Finish;
    end Reader2;
    task Writer is
        entry Finish;
    end Writer;

    -- Variable definition
    Start_Time1 : CALENDAR.time;
    Start_Time2 : CALENDAR.time;
    Start_Time3 : CALENDAR.time;
    Finish_Time1 : CALENDAR.time;
    Finish_Time2 : CALENDAR.time;
    Finish_Time3 : CALENDAR.time;
    Result1      : duration;
    Result2      : duration;
    Result3      : duration;
    Addr_List1   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
    Addr_List2   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
    Addr_List3   : RCDT.Pos_List_Type(1..ATTR_SIZE) := (0, others=>0);
    Attr_List1   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
    Attr_List2   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
    Attr_List3   : RCDT.Attr_List_Type(1..ATTR_SIZE) := ((0,200), others=>(0,200));
    Length1      : integer := 1;
    Length2      : integer := 1;
    Length3      : integer := 1;
    Outfile      : file_type;

    -- The body of task reader1
    task body Reader1 is
    begin
        Start_Time1 := CALENDAR.clock;
        for I in 1..NUMBER_OF_TIMES1 loop
            ROBCP.Read_Attrs(Addr_List1, Length1, Attr_List1);
        end loop;
        Finish_Time1 := CALENDAR.clock;
        Result1      := Finish_Time1 - Start_Time1;

```

```

    accept Finish;
exception
    when others =>
        put_line("Task Reader1 has an exception.");
end Reader1;

-- The body of task reader2
task body Reader2 is
begin
    Start_Time2 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES2 loop
        RODBCP.Read_Attrs(Addr_List2, Length2, Attr_List2);
    end loop;
    Finish_Time2 := CALENDAR.clock;
    Result2      := Finish_Time2 - Start_Time2;

    accept Finish;
exception
    when others =>
        put_line("Task Reader2 has an exception.");
end Reader2;

-- The body of task writer
task body Writer is
begin
    Start_Time3 := CALENDAR.clock;
    for I in 1..NUMBER_OF_TIMES3 loop
        RODBCP.Write_Attrs(Addr_List3, Length3, Attr_List3);
    end loop;
    Finish_Time3 := CALENDAR.clock;
    Result3      := Finish_Time3 - Start_Time3;

    accept Finish;
exception
    when others =>
        put_line("Task Writer has an exception.");
end Writer;

begin

    -- Terminate gracefully
    Reader1.Finish;
    Reader2.Finish;
    Writer.Finish;

    -- Write out the results
    create(Outfile, out_file, "rodbcomp1.out",
        form=>"world=>read, owner=>read_write");
    put_line(Outfile, "Task      Number_Of_Iterations      Times");
    put(Outfile, "Reader1      ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES1);
    FIX_IO.put(Outfile, Result1);
    new_line(Outfile);
    put(Outfile, "Reader2      ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES2);
    FIX_IO.put(Outfile, Result2);
    new_line(Outfile);
    put(Outfile, "Writer      ");
    INT_IO.put(Outfile, NUMBER_OF_TIMES3);
    FIX_IO.put(Outfile, Result3);

```

```
new_line(Outfile);  
close(Outfile);  
  
end RodbtstF1;
```

```

      t
      t
r rrr  oooo  oooo  ttttt
rr  r  o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t
r      o  o  o  o  t  t
r      oooo  oooo  tt

```

```

      d  b
      d  b
      d  b
r rrr  oooo  ddd d  b bbb  cccc  oooo  m m mm  p ppp
rr  r  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p
r      o  o  d  d  b  b  b  c  o  o  m m m  p  p
r      o  o  d  d  b  b  b  c  o  o  m m m  pp  p
r      o  o  d  dd  bb  b  c  c  o  o  m m m  p ppp
r      oooo  ddd d  b bbb  cccc  oooo  m m m  p ppp
                                     p
                                     p
                                     p

```

1  
11  
1 1  
1  
1  
1  
1  
1111

Job: rodbcomp1.dat  
Date: Wed Apr 15 19:32:46 1992

10  
0 100  
1 200  
2 300  
3 400  
4 500  
5 600  
6 700  
7 800  
8 900  
9 1000

10  
0 A  
1 B  
2 C  
3 D  
4 E  
5 F  
6 G  
7 H  
8 I  
9 J

10  
0 false  
1 false  
2 false  
3 false  
4 false  
5 false  
6 false  
7 false  
8 false  
9 false

10  
0 100.0  
1 200.0  
2 300.0  
3 400.0  
4 500.0  
5 600.0  
6 700.0  
7 800.0  
8 900.0  
9 1000.0



[illegible]

```
Job:  rodbcomp11.in
Date:  Wed Apr 15 19:29:58 1992
```

2500 2500 5000

r rrr	oooo	oooo	t t tttt
rr r	o o	o o	t
r	o o	o o	t
r	o o	o o	t
r	o o	o o	t t
r	oooo	oooo	tt

			d	b						1
			d	b						11
			d	b						1 1
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp			1
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p			1
r	o o	d d	b b	c	o o	m m m	p p			1
r	o o	d d	b b	c	o o	m m m	p p			1
r	o o	d dd	bb b	c c	o o	m m m	pp p			1
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp			11111
							p			
							p			
							p			

Job: rodbcomp11.out  
Date: Wed Apr 15 19:30:08 1992

Task	Number_Of_Iterations	Times
Reader1	2500	2.12396
Reader2	2500	1.71387
Writer	5000	2.13214

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr  oooo  oooo
rr  r  o  o  o  o
r      o  o  o  o
r      o  o  o  o
r      o  o  o  o
r      oooo  oooo

```

```

      d  b
      d  b
      d  b
      d  b bbb
      d  dd  b bbb  cccc  oooo  m m mm  p ppp
rr rrr  o  o  d  dd  bb  b  c  c  o  o  mm m m  pp  p
rr  r  o  o  d  d  b  b  c  c  o  o  m  m  m  p  p
r      o  o  d  d  b  b  c  c  o  o  m  m  m  p  p
r      o  o  d  dd  bb  b  c  c  o  o  m  m  m  pp  p
r      oooo  ddd d  b bbb  cccc  oooo  m  m  m  p ppp
                                     p
                                     p
                                     p

```

1  
11  
1 1  
1  
1  
1  
1  
11111

Job: rodbcomp12.in  
Date: Wed Apr 15 19:31:08 1992

5000 5000 5000

```

      t
      t
r rrr      oooo      oooo      ttttt
rr      r  o      o  o      o      t
r      o      o      o      t
r      o      o      o      t
r      o      o      o      t
r      oooo      oooo      tt

```

```

                        d      b
                        d      b
                        d      b
r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr      r  o      o  d      dd      bb      b      c      c      o      o      mm m m      pp      p
r      o      o      d      d      b      b      c      c      o      o      m m m      p      p
r      o      o      d      d      b      b      c      c      o      o      m m m      pp      p
r      o      o      d      dd      bb      b      c      c      o      o      m m m      p ppp
r      oooo      ddd d      b bbb      cccc      oooo      m m m      p ppp
                                           p
                                           p
                                           p

```

1  
11  
1 1  
1  
1  
1  
1  
1  
1111

Job: rodbcomp12.out  
Date: Wed Apr 15 19:31:23 1992

Task	Number_Of_Iterations	Times
Reader1	5000	4.06671
Reader2	5000	3.75201
Writer	5000	3.30518



11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b	i	n n
1	i	bb b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b				
		d	b				
		d	b				
r rrr	oooo	ddd d	b bbb	cccc	oooo	m m mm	p ppp
rr r	o o	d dd	bb b	c c	o o	mm m m	pp p
r	o o	d d	b b	c c	o o	m m m	p p
r	o o	d d	b b	c c	o o	m m m	pp p
r	oooo	ddd d	b bbb	cccc	oooo	m m m	p ppp
							p
							p
							p

---

Job: rodb\_component\_data\_types\_.ada  
Date: Wed Apr 15 19:25:00 1992

```

-- This package provides the constants, instantiated packages, system calls
-- and C functions interfaces to C language for RODB COMPONENT package.
with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
package RODB_COMPONENT_DATA_TYPES is

  -- Constants
  INT_SIZE      : constant integer := 10;
  CHAR_SIZE     : constant integer := 10;
  BOOL_SIZE     : constant integer := 10;
  FLT_SIZE      : constant integer := 10;
  SHMKEY        : constant integer := 99;
  SEMKEY        : constant integer := 100;
  SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
  CHAR_OFFSET   : constant integer := INT_SIZE*4;
  BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
  FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

  -- Data types
  type Attr_Type (Type_ID : integer := 0) is record
    case Type_ID is
      when 0 =>
        Int_Value : integer;
      when 1 =>
        Char_Value : character;
      when 2 =>
        Bool_Value : boolean;
      when 3 =>
        Flt_Value  : float;
      when others =>
        null;
    end case;
  end record;
  type Attr_List_Type is array(integer range <>) of Attr_Type;
  type Pos_List_Type is array(integer range <>) of integer;

  -- Package instantiation
  package INT_IO is new TEXT_IO.INTEGER_IO(integer);
  package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
  package FLT_IO is new TEXT_IO.FLOAT_IO(float);
  function FINT is new system.fetch_from_address(integer);
  function FCHAR is new system.fetch_from_address(character);
  function FBOOL is new system.fetch_from_address(boolean);
  function FFLT is new system.fetch_from_address(float);
  procedure AINT is new system.assign_to_address(integer);
  procedure ACHAR is new system.assign_to_address(character);
  procedure ABOOL is new system.assign_to_address(boolean);
  procedure AFLT is new system.assign_to_address(float);

  -- Shared memory system call interface
  function SHMGET(KEY : in integer;
                 SIZE : in integer;
                 FLAG : in integer) return integer;
  pragma INTERFACE(C, SHMGET);
  pragma INTERFACE_NAME(SHMGET, "shmget");
  function SHMAT(SHMID : in integer;
                SHMADDR : in system.address;
                FLAG : in integer) return system.address;
  pragma INTERFACE(C, SHMAT);
  pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD : in integer;
                BUFF : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

-- Semaphore system call and C function interface
function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function SEMPRINT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMPRINT);
pragma INTERFACE_NAME(SEMPRINT, "semprint");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

end RODB_Component_Data_Types;

```

ll		b		
l	i	b	i	
l		b		
l	ii	b bbb	ii	n nnn
l	i	bb b	i	nn n
l	i	b b	i	n n
l	i	b b	i	n n
l	i	bb b	i	n n
lll	iii	b bbb	iii	n n

			d	b			
			d	b			
			d	b			
r rrr	oooo	ddd d	b bbb		cccc	oooo	m m mm
rr r	o o	d dd	bb b		c c	o o	mm m m
r	o o	d d	b b		c c	o o	m m m
r	o o	d dd	bb b		c c	o o	m m m
r	oooo	ddd d	b bbb		cccc	oooo	m m m
							p ppp
							pp p
							p p
							pp p
							p ppp
							p
							p
							p

---

Job: rodb\_component\_ada  
Date: Wed Apr 15 19:25:20 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

  -- Package renaming
  package RCDT renames Rodb_Component_Data_Types;

  -- Exception definition
  Shm_Exception : exception;
  Shm_Outrange  : exception;
  Sem_Exception : exception;

  -- Read attributes from RODB components
  procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                       Length     : in      integer;
                       Attr_List  : in out RCDT.Attr_List_Type);

  -- Write attributes to RODB components
  procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length     : in      integer;
                        Attr_List  : in      RCDT.Attr_List_Type);

  -- Print out the semaphore values
  procedure Print_Sems;

  -- Load RODB components from a disk file
  procedure Load_Comps(Filename : in      string);

  -- Save RODB components to a disk file
  procedure Save_Comps(Filename : in      string);

  -- Shutdown the RODB components
  procedure Shutdown_Comps;

end Rodb_Component;

```

```

r rrr      oooo      ddd d      b bbb      cccc      oooo      m m mm      p ppp
rr   r     o   o     d   dd     bb   b     c   c     o   o     mm m   m   pp   p
r        o   o     d   d     b   b     c       o   o     m   m   m   p   p
r        o   o     d   d     b   b     c       o   o     m   m   m   p   p
r        o   o     d   dd     bb   b     c       o   o     m   m   m   pp  p
r          oooo      ddd d      b bbb      cccc      oooo      m   m   m   p ppp

```

---

Job: rodb\_component.ada  
Date: Wed Apr 15 19:25:36 1992

```

with TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, PREEMPTION_CONTROL, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is

```

```

    Temp : system.address;

```

```

    Flag : integer;

```

```

begin

```

```

    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;

```

```

    Flag := RCDT.READBEG(Semid);

```

```

    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;

```

```

    if Flag = -1 then

```

```

        raise Sem_Exception;

```

```

    end if;

```

```

    for I in 1..Length loop

```

```

        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;

```

```

        end if;

```

```

        Temp := Shmaddr + system.offset(Addr_List(I));

```

```

        if (Addr_List(I) < RCDT.CHAR_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));

```

```

        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));

```

```

        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then

```

```

            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));

```

```

        else

```

```

            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));

```

```

        end if;

```

```

    end loop;

```

```

    -- delay 10.0;

```

```

    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;

```

```

    Flag := RCDT.READEND(Semid);

```

```

    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;

```

```

    if Flag = -1 then

```

```

        raise Sem_Exception;

```

```

    end if;

```

```

end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is

```

```

    Temp : system.address;
    Flag : integer;
begin
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.WRITEBEG(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1.. Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            RCDT.AINT(Temp, Attr_List(I).Int_Value);
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
        else
            RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
        end if;
    end loop;
    -- delay 10.0;
    -- PREEMPTION_CONTROL.DISABLE_PREEMPTION;
    Flag := RCDT.WRITEEND(Semid);
    -- PREEMPTION_CONTROL.ENABLE_PREEMPTION;
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Write_Attrs;

-- Print out the semaphore values
procedure Print_Sems is
    Flag : integer;
begin
    Flag := RCDT.SEMPRINT(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Print_Sems;

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--     Number_Of_Integers
--     Position1 Integer1
--     Position2 Integer2
--     ...
--     Number_Of_Characters
--     Position1 Character1
--     Position2 Character2
--     ...
--     Number_Of_Booleans
--     Position1 Boolean1
--     Position2 Boolean2
--     ...
--     Number_Of_Floats
--     Position1 Float1
--     Position2 Float2

```



```

--      ...
procedure Load_Comps(Filename : in      string) is
  Infile : FILE_TYPE;
  Temp   : system.address;
  Flag   : integer;
begin
  open(Infile, in_file, Filename);

  -- Initialize RODB Integer Component
  for I in 1..RCDT.INT_SIZE loop
    Temp := Shmaddr + system.offset((I-1)*4);
    RCDT.AINT(Temp, 0);
  end loop;
  Load_Ints(Infile);

  -- Initialize RODB Character Component
  for I in 1..RCDT.CHAR_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
  end loop;
  Load_Chars(Infile);

  -- Initialize RODB Boolean Component
  for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
  end loop;
  Load_Bools(Infile);

  -- Initialize RODB Float Component
  for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
  end loop;
  Load_Flts(Infile);

  close(Infile);

  Flag := RCDT.SEMSINIT(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;

exception
  when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
  when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
  when Sem_Exception =>
    put_line("Semaphore cannot be initialized.");
    raise Sem_Exception;
  when others =>
    put_line("Unknown exception.");
    put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file

```

```

-- The structure of the disk file is as following:
--   Number_Of_Integers
--   Position1 Integer1
--   Position2 Integer2
--   ...
--   Number_Of_Characters
--   Position1 Character1
--   Position2 Character2
--   ...
--   Number_Of_Booleans
--   Position1 Boolean1
--   Position2 Boolean2
--   ...
--   Number_Of_Floats
--   Position1 Float1
--   Position2 Float2
--   ...
procedure Save_Comps(Filename : in string) is
  Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
          form=>"world=>read, owner=>read_write");
    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint_error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```

pragma page;

-- Load all the integers from a disk file to RODB Integer Component  
procedure Load\_Ints(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Int    : integer;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
    RCDT.AINT(Temp_Addr, Temp_Int);
  end loop;
end Load_Ints;
```

-- Load all the characters from a disk file to RODB Character Component  
procedure Load\_Chars(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Char   : character;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    get(Infile, Temp_Char);           -- Skip a space
    get(Infile, Temp_Char);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
      raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
    RCDT.ACHAR(Temp_Addr, Temp_Char);
  end loop;
end Load_Chars;
```

-- Load all the booleans from a disk file to RODB Boolean Component  
procedure Load\_Bools(Infile : in FILE\_TYPE) is

```
  Length      : Integer;
  Temp_Pos    : integer;
  Temp_Bool   : boolean;
  Temp_Addr   : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    BOOL_IO.get(Infile, Temp_Bool);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
```

```

        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
    RCDT.ABOOL(Temp_Addr, Temp_Bool);
end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in    FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        FLT_IO.get(Infile, Temp_Flt);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
        RCDT.AFLT(Temp_Addr, Temp_Flt);
    end loop;
end Load_Flts;

pragma page;

-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in    FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
    end loop;
end Save_Chars;

```

```

        new_line(Outfile);
    end loop;
end Save_Chars;

-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;

    -- Initialize the RODB Components
    Load_Comps("rodbcomp.dat");
end Rodb_Component;

```

11		b		
1	i	b	i	
1		b		
1	ii	b bbb	ii	n nnn
1	i	bb b	i	nn n
1	i	b b b	i	n n
1	i	bb b	i	n n
111	iii	b bbb	iii	n n

		d	b			
		d	b			
		d	b			
r rrr	oooo	ddd d	b bbb			
rr r	o o	d dd	bb b			
r	o o	d d	b b			
r	o o	d dd	bb b			
r	oooo	ddd d	b bbb			

t tttt	eeee	ssss		t tttt		
t	e e	s s		t		
t	eeeeee	ss		t		
t	e	ss		t		
t t	e e	s s		t t		
tt	eeee	ssss		tt		

Job: rodb\_test\_data1.ada  
Date: Wed Apr 15 19:26:35 1992

```

with TEXT_IO;
use TEXT_IO;
package RODB_Test_Data1 is

    Number_Of_Times1 : integer;
    Number_Of_Times2 : integer;
    Number_Of_Times3 : integer;

    package INT_IO is new TEXT_IO.INTEGER_IO(integer);

end RODB_Test_Data1;

with TEXT_IO;
use TEXT_IO;
package body RODB_Test_Data1 is
    Infile : file_type;
begin
    open(Infile, in_file, "rodbcompl.in");
    INT_IO.get(Infile, Number_Of_Times1);
    INT_IO.get(Infile, Number_Of_Times2);
    INT_IO.get(Infile, Number_Of_Times3);
    close(Infile);
end RODB_Test_Data1;

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
r
r
      oooo
      o
      o
      o
      o
      o
      o
      oooo
      oooo
      oooo

```

```

      i
      i
      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      i
      i
      i
      i
      i
      iii

      n nnn
      nn  n
      n    n
      n    n
      n    n
      n    n

      i
      i
      i
      i
      i
      iii

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

      ..
      ..

```

Job: semsinit.c  
 Date: Wed Apr 15 19:35:49 1992



```
/*File: semsinit.c This is semaphore init subroutine to initialize semaphores*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

  t
  t
ttttt
  t
  t
  t
  t  t
    tt

```

```

      CCCC
    C      C
    C
    C
    C      C
      CCCC

```

```
Job:  readbeg.c
Date:  Wed Apr 15 19:33:56 1992
```

```

/* File: readbeg.c This is read begin subroutine to set reading protection */
/* Protection now disabled for this test */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf RREAD_START      = { 0, 1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1, 0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2, 0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[5]; /* Three semaphore operations */
    int flag;
    void perror();
    tid_t my_tid;
    int my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = gettid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */ /* Disable prevention of preemption */

    /* Perform three semaphore operations */ /* Disable semaphores also */
    /* one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[2] = RWAIT_NO_WRITE_LOCK; */ /* Wait for no more writer */
    /* one_of_n_readers[3] = RWAIT_NO_WRITE_DESIRE; */ /* Wait for no more writer */
    /* one_of_n_readers[4] = RREAD_START; */ /* Prevent writer in */
    /* flag = semop(semid, one_of_n_readers, 5); */ /* Lock the critical section */
    /* if (flag == -1) {
        perror("readbeg fails: ");
    } */

    /* Lower the priority to the normal */ /* Now, no need to reenab*/
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

```

r rrr      eeee      aaaa      ddd d      eeee      n nnn      ddd d      cccc
rr  r      e   e      a      d   dd      e   e      nn   n      d   dd      c   c
r         eeeee     aaaaa      d   d      eeeee     n   n      d   d      c   c
r         e   e      a   a      d   d      e   e      n   n      d   d      c   c
r         e   e      a   aa     d   dd     e   e      n   n      d   dd     ..  c   c
r         eeee      aaaa a      ddd d      eeee      n   n      ddd d      ..  cccc

```

Job: readend.c  
Date: Wed Apr 15 19:34:02 1992

```

/* File: readend.c This is read end subroutine to reset protection */
/* Action now disabled this is a dummy program */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int    flag;
    void    perror();
    tid_t my_tid;
    int    my_prio;

    /* Raise the priority to prevent the preemption */ /* Now disabled */
    my_tid = getstid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */ /* Commented out */

    /* Perform the semaphore operation */ /* Now no need */
    /* flag = semop(semid, &RREAD_END, 1); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("readend fails: ");
    } */

    /* Lower the priority to the normal */
    /* fast_setprio(my_tid, my_prio); */ /* Now no need since not raised */

    return flag;
}

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
r

o o o o
o   o
o   o
o   o
o   o
o o o o

o o o o
o   o
o   o
o   o
o   o
o o o o


```

```

      i
      t
      t
      ttttt
      t
      t
      t
      t t
      tt

w      w  r rrr
w w w rr  r
w w w r
w w w r
w w w r
ww ww  r

i i
i
i
i
i
i i i

e e e e
e   e
e e e e e
e   e
e   e
e e e e

b
b
b
b bbb
bb  b
b   b
bb  b
b bbb

e e e e
e   e
e e e e e
e   e
e   e
e e e e

g g g g
g   g g
g   g
g   g
g   g g
g g g g
g   g
g g g g

..
..


```

Job: writebeg.c  
Date: Wed Apr 15 19:34:09 1992

```

/* File: writebeg.c This is write begin subroutine to set protection */
/* Now protection disabled, this is dummy function */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/sched.h>
#include <st.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START = { 0, 1, 0};
struct sembuf WWRITE_LOCK = { 1, 1, 0};
struct sembuf WWRITE_DESIRE = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int flag;
    void perror();
    tid_t my_tid;
    int my_prio;

    /* Raise the priority to prevent the preemption */ /* Not anymore !! */
    my_tid = gettid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */ /* No prevention of preemption */

    /* Make write request by doing a semaphore operation */ /* Not now !! */
    /* flag = semop(semid, &WWRITE_DESIRE, 1); */
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    } /* /* This program now does nothing constructive */

    /* Perform four semaphore operations */ /* Disabled !! */
    /* sole_writer[0] = WWAIT_NO_READERS; */ /* Wait for no more readers */
    /* sole_writer[1] = WWRITE_LOCK; */ /* preventing succeeding readers */
    /* sole_writer[2] = WREAD_START; */ /* preventing succeeding writers */
    /* sole_writer[3] = WIN_PROGRESS_WRITE; */ /* Cancel the write-request */
    /* flag = semop(semid, sole_writer, 4); */ /* Lock the critical section */
    /* if (flag == -1) { */
        perror("Write_Start in writebeg fails: ");
    } /*

    /* Lower the priority to the normal */ /* Not necessary ! */
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

```

      t
      t
r rrr      oooo      oooo      ttttt
rr  r      o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t
r          o  o      o  o      t  t
r          oooo      oooo      tt

```

```

              i          t          d
              ii         t         d
w      w  r rrr      ttttt      eeee      eeee      n nnn      ddd d
w  w  w  rr  r      t          e   e      e   e      nn  n      d  dd
w  w  w  r          t          eeeeee     eeeeee     n   n      d  d
w  w  w  r          t          e          e          n   n      d  d
w  w  w  r          t  t       e   e      e   e      n   n      d  dd
  ww ww  r      iii         tt      eeee      eeee      n   n      ddd d

```

Job: writeend.c  
Date: Wed Apr 15 19:34:20 1992



```

/* File: writeend.c This is write end subroutine to reset write protection */
/* Now protection disabled for this test */
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>
#include      <sys/sched.h>
#include      <st.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();
    tid_t   my_tid;
    int     my_prio;

    /* Raise the priority to prevent the preemption */
    my_tid = getstid();
    my_prio = getprio(getpid());
    /* fast_setprio(my_tid, 31); */ /* Disabled */

    /* Perform the semaphore operation */ /* Now no semops !! */
    /* sole_writer[0] = WREAD_END; */ /* Allow writer in */
    /* sole_writer[1] = WWRITE_UNLOCK; */ /* Allow reader in */
    /* flag = semop(semid, sole_writer, 2); */ /* Unlock critical section */
    /* if (flag == -1) {
        perror("writeend fails: ");
    } */

    /* Lower the priority to the normal */ /* No need now !! */
    /* fast_setprio(my_tid, my_prio); */

    return flag;
}

```

```

      t
      t
      ttttt
      t
      t
      t
      t t
      tt

r rrr
rr  r
r
r
r
r
r

  oooo      oooo
o   o      o   o
o   o      o   o
o   o      o   o
o   o      o   o
  oooo      oooo

```

```

ssss      eeee      m m mm      ssss      r rrr      m m mm      v      v      cccc
s    s    e    e    mm m m    s    s    rr  r    mm m m    v      v    c
ss      eeeeeee  m m m    ss      r      m m m    v      v    c
      ss      e    e    m m m    ss      r      m m m    v      v    c
s    s    e    e    m m m    s    s    r      m m m    v v      ..
ssss      eeee      m m m    ssss      r      m m m    v      ..    cccc

```

Job: sarmsv.c  
 Date: Wed Apr 15 19:35:41 1992

```
/* File: semsrmv.c This is semaphore remove subroutine to remove semaphores */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semsrmv fails: ");
    }
    return(flag);
}
```



## **Appendix E**

### **Concurrent (Competing) Reads and Writes**

**Code for demonstrating the lack of mutual exclusion in a critical section for reading/writing to a simulated RODB component. The lack of mutual exclusion is presumably caused by the non-atomic nature of the semop system-call algorithm for an array of semaphores.**



```

      t
      t
rrr   oooo   oooo   ttttt
  r   o   o   o   o   t
      o   o   o   o   t
      o   o   o   o   t
      o   o   o   o   t
      oooo   oooo   tt

```

```

      d
      d
      d
rrr   eeee   aaaa   ddd d
:  r   e   e   a   d   dd
      e   e   a   d   d
      eeeee   aaaaa   d   d
      e   a   a   d   d
      e   e   a   aa   d   dd
      eeee   aaaa a   ddd d

```

```

m m mm   eeee
mm m m   e   e
m m m   eeeee
m m m   e
m m m   e   e
m m m   eeee

```

```

..
..

```

Job: read.me  
Date: Tue Apr 14 22:24:53 1992

This directory stores all the files to build up an RODB "attribute" component. The protection mechanism is that locking is set at the RODB level. During the lock setting, there is NO prevention of preemption. WE ASSUME A SYSTEM CALL IS AN ATOMIC ACTION. Assume there is one set of three UNIX semaphores in the whole system. Before actual reading, a set of THREE semaphore operations will be imposed on the semaphores. After actual reading one semaphore array operation will be imposed on the semaphores. Before actual writing, there are two levels of operations:

write-intent on one level and write-lock and read-lock on the other.

For write-intent only one semaphore operation will be imposed on the semaphore and for write-lock a set of four semaphore operations will be ordinarily imposed on the semaphores (i.e. including test of read-lock semaphore increase read-lock, set write-lock and clear write-intent semaphore).

BUT ON LYNX SEMAPHORE OPERATIONS, IF A PROCESS EXECUTING SEMOP IS PREEMPTED OR EVEN SLEEPS ON A SEMAPHORE EVENT, THERE IS NO GUARANTEE THAT SEMOP WILL RESUME AT THE VERY BEGINNING OF THE SEMOP ALGORITHM.

After actual writing, a set of two release-semaphore operations will ordinarily be imposed on the semaphores.

IN THIS PROGRAM, A PAUSE HAS BEEN INTRODUCED IN THE WRITER "CODE" SO THAT THESE TWO OPERATIONS (VIZ. WRITER LOCK AND READER LOCK) ARE NOT RELEASED. WHAT THIS TEST SHOWS IS FACT THAT IF THE READER HAS SLEPT ON THE WRITER INTENT SEMAPHORE THAT HAD BEEN ACTIVATED BY THE WRITER AND NOW HAD BEEN RELEASED AND THE READER NOW IS AWAKENED, IT SHOULD, BUT WILL NOT GO BACK TO THE BEGINNING OF THE ALGORITHM TO CHECK THE STATUS OF THE WRITER LOCK SEMAPHORE. INSTEAD IT RESUMES WHERE IT LEFT OFF AND ONLY CHECKS THE WRITER INTENT SEMAPHORE WHICH HAS JUST BEEN CLEARED. HENCE, WE NOW HAVE BOTH A READER AND A WRITER IN THEIR CRITICAL SECTIONS AT THE SAME TIME.

To run this program --once compiled-- three terminals are required. Each terminal should log into the same account and execute rodb test7. When this is done a menu will appear for each process on its respective terminal screen. One process should choose option "4" to load the shared memory and press the enter key (i.e. <CR>). The question is then "asked" for the name of the file from which to load the contents of the shared memory. The simplest is to press "Enter" since there is a default data file which will be loaded by pressing the Enter key (i.e. <CR>). Once this is done the memory may be viewed by selecting option 3. Then, when the menu appears again, this first terminal should choose the option "1" to read a list of attributes followed by <CR>. When the question is asked "how many" the simplest answer is to type 1 <CR>. The question will then be raised as to the address to be "read", the simplest answer is to type the number "0" (without the quotes) BUT DO NOT PRESS "Enter". Then set up the next terminal by running another copy of rodb\_test7 and choose the option "2" to write a list of attributes <CR>. Then, when the question is "asked" for "how many", the answer is to type 1 <CR>. As before the address "question" will be asked. Type in the number "0" as before followed by <CR>. The question will then be asked for the value to be written. You may type some integer such as 200 (the number 100 is the default at address 0). Again DO NOT TYPE <CR> or Enter !! Then start the third process from the THIRD terminal. This time choose the option "1" again to read. Again select one (1) attribute and again select address "0" BUT DO NOT TYPE <CR>. NOW, go back to terminal one (1) and press Enter (<CR>). Right after that (less than five seconds) press Enter (<CR>) on terminal two (2). After that (less than five but more than two seconds) press "Enter" on terminal three (3). Clearly the first process will encounter the



semaphores before process two and will lock out process two since it will increase the reader semaphore. Process two (2) will set the write-intent semaphore but will block on the reader semaphore. Process three (3) will set the write-lock semaphore and find it NOT set (Process one (1) is structured to stay in the "reader" critical section about ten seconds so a writer will be "locked out"). HOWEVER, Process three (3) will find the write-intent semaphore set so it will "sleep" on the event that the write-intent semaphore is "cleared". Meanwhile eventually -- after ten seconds -- the first reader finishes and "clears" the reader semaphore. Immediately the the second process (2) will wake to find the reader semaphore be zero. This writer process (2) will then set the "write-lock" semaphore and will release the write-intent semaphore so as to allow higher priority readers to wake up and sleep on the "write-lock" semaphore rather than the write-intent semaphore (2) then enters the critical section to access the DB Component. THIS IS WHERE THE PROBLEM APPEARS since process three (3) apparently does wake up BUT DOES NOT "GO TO START" and recheck the "write\_lock" semaphore and the "write-intent" semaphore but rather just continues on (perhaps) not checking the "write-lock" semaphore but only the "write-intent" semaphore and finding it "clear" AND then it increases the "read-lock" semaphore and reads the shared memory. It is thus reading the shared memory at essentially shared memory. You will see the process three write out what it has read and terminate. Process two (2), which started before process three (3) is locked at its "pause" statement just before accessing the shared memory. Process three (3) finishes even though it should be blocked from entering the "write\_lock" semaphore set by process two (2).

Since the Lynx semop algorithm does not meet the System V requirements and further even if it did in the non-preemptive case, it might fail under the real-time preemptive case. That is to say, if the process does not sleep but is preempted during the execution of algorithm semop when it resumed it would not know that it had been preempted and would continue on. Thus the same scenario could happen as above even if the reader process did not sleep but was preempted by an high-priority writer just before it increased the "read-lock" semaphore.

```

      t
      t
r rrr      oooo      oooo      ttttt
rr   r      o   o      o   o      t
r          o   o      o   o      t
r          o   o      o   o      t t
r          o   o      o   o      tt
r          oooo      oooo

```

```

      d   b
      d   b
      d   b
r rrr      oooo      ddd d   b bbb
rr   r      o   o      d   dd  bb  b
r          o   o      d   d   b   b
r          o   o      d   d   b   b
r          o   o      d   dd  bb  b
r          oooo      ddd d   b bbb

```

```

      t
      t
      ttttt      eeee      ssss      ttttt
      t          e   e      s   s      t
      t          eeeee      ss        t
      t          e   e      ss        t
      t t        e   e      s   s      t t
      tt         eeee      ssss      tt

```

---

Job: rodb\_test7.ada  
 Date: Tue Apr 14 22:25:16 1992



```
-- Output a list of Attributes according to their addresses
procedure Output_Attr_List(Attr_List : in      RCDT.Attr_List_Type;
                           Length      : in      integer) is
```

```
begin
```

```
  for I in 1.. Length loop
    put("Attribute number");
    INT_IO.put(I, width => 3);
    put(" is ");
    case Attr_List(I).Type_ID is
      when 0 =>
        put("Integer: ");
        INT_IO.put(Attr_List(I).Int_Value);
      when 1 =>
        put("Character: ");
        put(Attr_List(I).Char_Value);
      when 2 =>
        put("Boolean: ");
        BOOL_IO.put(Attr_List(I).Bool_Value);
      when 3 =>
        put("Float: ");
        FLT_IO.put(Attr_List(I).Flt_Value);
      when others =>
        null;
    end case;
    new_line;
  end loop;
end Output_Attr_List;
```

```
begin
```

```
  loop
```

```
    put_line("1----Read a list of attributes");
    put_line("2----Write a list of attributes");
    put_line("3----print out the shared memory");
    put_line("4----Load the shared memory");
    put_line("0----Exit");
    put("Input your selection: ");
    INT_IO.get(Choice);
    skip_line;
    case Choice is
      when 0 =>
        exit;
      when 1 =>
        put("How many attributes do you want: ");
        INT_IO.get(Length);
        Input_Addr_List(Addr_List, Length);
        RODBCP.Read_Attrs(Addr_List, Length, Attr_List);
        Output_Attr_List(Attr_List, Length);
      when 2 =>
        put("How many attributes do you want: ");
        INT_IO.get(Length);
        Input_Addr_List(Addr_List, Length);
        Input_Attr_List(Attr_List, Length, Addr_List);
        RODBCP.Write_Attrs(Addr_List, Length, Attr_List);
      when 3 =>
        put("Enter the filename to send to(none to screen): ");
        get_line(Filename, File_Len);
        RODBCP.Save_Comps(Filename(1..File_Len));
      when 4 =>
        put("Enter the filename to load from(none from rodbcomp.dat): ");
```

```

    get_line(Filename, File_Len);
    if (File_Len /= 0) then
        RODBCP.Load_Comps(Filename(1..File_Len));
    else
        RODBCP.Load_Comps("rodbcomp.dat");
    end if;
    when others =>
        put_line("Input error!");
    end case;
end loop;
RODBCP.Shutdown_Comps;
ception
when Shm_Exception =>
    put_line("Shared memory not accessible.");
when Shm_Outrange =>
    put_line("Shared memory out of range.");
when Sem_Exception =>
    put_line("Semaphores not accessible.");
d Rodb_Test7;

```

r rrr	oooo	oooo	t
rr r	o o	o o	t
r	o o	o o	ttttt
r	o o	o o	t
r	o o	o o	t
r	oooo	oooo	t t
			tt

r rrr	oooo	ddd d	d	b
rr r	o o	d dd	d	b
r	o o	d d	d	b
r	o o	d d	b bbb	b bbb
r	o o	d dd	bb b	b b
r	oooo	ddd d	b b	b b
			bb b	b b
			b bbb	b bbb

cccc	oooo	m m mm	p ppp
c c	o o	mm m m	pp l
c c	o o	m m m	p l
c c	o o	m m m	p l
cccc	oooo	m m m	pp l
		m m m	p ppp
			p
			p
			p

Job: rodb\_component\_data\_types\_.ada  
Date: Tue Apr 14 22:25:41 1992

This package provides the constants, instantiated packages, system calls and C functions interfaces to C language for RODB COMPONENT package.

```

th TEXT_IO, SYSTEM;
e TEXT_IO, SYSTEM;
ckage RODB_COMPONENT_DATA_TYPES is

```

```

-- Constants
INT_SIZE      : constant integer := 10;
CHAR_SIZE     : constant integer := 10;
BOOL_SIZE     : constant integer := 10;
FLT_SIZE      : constant integer := 10;
SHMKEY        : constant integer := 99;
SEMKEY        : constant integer := 100;
SHM_SIZE      : constant integer := INT_SIZE*4+CHAR_SIZE+BOOL_SIZE+4*FLT_SIZE;
CHAR_OFFSET   : constant integer := INT_SIZE*4;
BOOL_OFFSET   : constant integer := CHAR_OFFSET + CHAR_SIZE*1;
FLT_OFFSET    : constant integer := BOOL_OFFSET + BOOL_SIZE*1;

```

```

-- Data types
type Attr_Type(Type_ID : integer := 0) is record
  case Type_ID is
    when 0 =>
      Int_Value : integer;
    when 1 =>
      Char_Value : character;
    when 2 =>
      Bool_Value : boolean;
    when 3 =>
      Flt_Value : float;
    when others =>
      null;
  end case;
end record;
type Attr_List_Type is array(integer range <>) of Attr_Type;
type Pos_List_Type is array(integer range <>) of integer;

```

```

-- Package instantiation
package INT_IO is new TEXT_IO.INTEGER_IO(integer);
package BOOL_IO is new TEXT_IO.ENUMERATION_IO(boolean);
package FLT_IO is new TEXT_IO.FLOAT_IO(float);
function FINT is new system.fetch_from_address(integer);
function FCHAR is new system.fetch_from_address(character);
function FBOOL is new system.fetch_from_address(boolean);
function FFLT is new system.fetch_from_address(float);
procedure AINT is new system.assign_to_address(integer);
procedure ACHAR is new system.assign_to_address(character);
procedure ABOOL is new system.assign_to_address(boolean);
procedure AFLT is new system.assign_to_address(float);

```

```

-- Shared memory system call interface
function SHMGET(KEY   : in integer;
                SIZE  : in integer;
                FLAG   : in integer) return integer;

pragma INTERFACE(C, SHMGET);
pragma INTERFACE_NAME(SHMGET, "shmget");
function SHMAT(SHMID   : in integer;
               SHMADDR  : in system.address;
               FLAG     : in integer) return system.address;

pragma INTERFACE(C, SHMAT);
pragma INTERFACE_NAME(SHMAT, "shmat");

```

```

function SHMDT(SHMADDR : in system.address) return integer;
pragma INTERFACE(C, SHMDT);
pragma INTERFACE_NAME(SHMDT, "shmdt");
function SHMCTL(SHMID : in integer;
                CMD    : in integer;
                BUFF   : in system.address) return integer;
pragma INTERFACE(C, SHMCTL);
pragma INTERFACE_NAME(SHMCTL, "shmctl");

-- Semaphore system call and C function interface
function SEMGET(KEY : in integer;
                NSEMS : in integer;
                FLAG  : in integer) return integer;
pragma INTERFACE(C, SEMGET);
pragma INTERFACE_NAME(SEMGET, "semget");
function SEMSINIT(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSINIT);
pragma INTERFACE_NAME(SEMSINIT, "semsinit");
function READBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, READBEG);
pragma INTERFACE_NAME(READBEG, "readbeg");
function READEND(SEMID : in integer) return integer;
pragma INTERFACE(C, READEND);
pragma INTERFACE_NAME(READEND, "readend");
function WRITEBEG(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEBEG);
pragma INTERFACE_NAME(WRITEBEG, "writebeg");
function WRITEEND(SEMID : in integer) return integer;
pragma INTERFACE(C, WRITEEND);
pragma INTERFACE_NAME(WRITEEND, "writeend");
function SEMSRMV(SEMID : in integer) return integer;
pragma INTERFACE(C, SEMSRMV);
pragma INTERFACE_NAME(SEMSRMV, "semsrmv");

end RODB_Component_Data_Types;

```



```

rrr      oooo  oooo  ttttt
  r      o  o  o  o  t
        o  o  o  o  t
        o  o  o  o  t
        o  o  o  o  t
        oooo  oooo  tt

```

```

rrr      d  b
  r      d  b
        d  b
        ddd d b bbb
        d dd bb  b
        d  d b  b
        d  d b  b
        d dd bb  b
        ddd d b bbb

```

```

cccc  oooo  m m mm  p ppp
c  c  o  o  mm m  m  pp  p
c  c  o  o  m  m  m  p  p
c  c  o  o  m  m  m  pp  p
cccc  oooo  m  m  m  p ppp
                               p
                               p
                               p

```

---

Job: rodb\_component\_.ada  
 Date: Tue Apr 14 22:25:57 1992

```

with TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, SYSTEM, Rodb_Component_Data_Types;
package Rodb_Component is

  -- Package renaming
  package RCDT renames Rodb_Component_Data_Types;

  -- Exception definition
  Shm_Exception : exception;
  Shm_Outrange  : exception;
  Sem_Exception : exception;

  -- Read attributes from RODB components
  procedure Read_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                       Length    : in      integer;
                       Attr_List : in out RCDT.Attr_List_Type);

  -- Write attributes to RODB components
  procedure Write_Attrs(Addr_List : in      RCDT.Pos_List_Type;
                        Length     : in      integer;
                        Attr_List  : in      RCDT.Attr_List_Type);

  -- Load RODB components from a disk file
  procedure Load_Comps(Filename : in      string);

  -- Save RODB components to a disk file
  procedure Save_Comps(Filename : in      string);

  -- Shutdown the RODB components
  procedure Shutdown_Comps;

end Rodb_Component;

```

```

rrr      oooo      oooo      t
r        o  o      o  o      t
        o  o      o  o      t
        o  o      o  o      t
        o  o      o  o      t
        oooo      oooo      tt

```

```

rrr      d      b
r        d      b
        d      b
        ddd d   b bbb
        d  dd  bb  b
        d  d   b    b
        d  d   b    b
        d  dd  bb  b
        ddd d   b bbb

```

```

cccc      oooo      m m mm      p ppp
c  c      o  o      mm m  m      pp  p
c  c      o  o      m  m  m      p   p
c  c      o  o      m  m  m      p   p
c  c      o  o      m  m  m      pp  p
cccc      oooo      m  m  m      p ppp
                                     p
                                     p
                                     p

```

---

Job: rodb\_component.ad  
 Date: Tue Apr 14 22:37:48 1992

```

with TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types;
use TEXT_IO, CALENDAR, SYSTEM, Rodb_Component_Data_Types;
package body Rodb_Component is

```

```

--Local variables

```

```

Shmid : integer;
Shmaddr : system.address;
Semid : integer;

```

```

-- Local subprograms

```

```

procedure Load_Ints(Infile : in FILE_TYPE);
procedure Load_Chars(Infile : in FILE_TYPE);
procedure Load_Bools(Infile : in FILE_TYPE);
procedure Load_Flts(Infile : in FILE_TYPE);
procedure Save_Ints(Outfile : in FILE_TYPE);
procedure Save_Chars(Outfile : in FILE_TYPE);
procedure Save_Bools(Outfile : in FILE_TYPE);
procedure Save_Flts(Outfile : in FILE_TYPE);

```

```

-- Read attributes from RODB components

```

```

procedure Read_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in out RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    Flag := RCDT.READBEG(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
    for I in 1..Length loop
        if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp := Shmaddr + system.offset(Addr_List(I));
        if (Addr_List(I) < RCDT.CHAR_OFFSET) then
            Attr_List(I) := (Type_ID => 0, Int_Value => RCDT.FINT(Temp));
        elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
            Attr_List(I) := (Type_ID => 1, Char_Value => RCDT.FCHAR(Temp));
        elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
            Attr_List(I) := (Type_ID => 2, Bool_Value => RCDT.FBOOL(Temp));
        else
            Attr_List(I) := (Type_ID => 3, Flt_Value => RCDT.FFLT(Temp));
        end if;
    end loop;
    delay 10.0;
    Flag := RCDT.READEND(Semid);
    if Flag = -1 then
        raise Sem_Exception;
    end if;
end Read_Attrs;

```

```

-- Write attributes to RODB components

```

```

procedure Write_Attrs(Addr_List : in RCDT.Pos_List_Type;
                    Length : in integer;
                    Attr_List : in RCDT.Attr_List_Type) is
    Temp : system.address;
    Flag : integer;
begin
    Flag := RCDT.WRITEBEG(Semid);

```

```

if Flag = -1 then
    raise Sem_Exception;
end if;
for I in 1.. Length loop
    if (Addr_List(I) < 0) or (Addr_List(I) > RCDT.SHM_SIZE-1) then
        raise Shm_Outrange;
    end if;
    Temp := Shmaddr + system.offset(Addr_List(I));
    if (Addr_List(I) < RCDT.CHAR_OFFSET) then
        RCDT.AINT(Temp, Attr_List(I).Int_Value);
    elsif (Addr_List(I) < RCDT.BOOL_OFFSET) then
        RCDT.ACHAR(Temp, Attr_List(I).Char_Value);
    elsif (Addr_List(I) < RCDT.FLT_OFFSET) then
        RCDT.ABOOL(Temp, Attr_List(I).Bool_Value);
    else
        RCDT.AFLT(Temp, Attr_List(I).Flt_Value);
    end if;
end loop;
delay 10.0;
Flag := RCDT.WRITEEND(Semid);
if Flag = -1 then
    raise Sem_Exception;
end if;
end Write_Attrs;

```

```

-- Load RODB Components from a disk file.
-- The structure of disk file is as following:
--      Number_Of_Integers
--      Position1   Integer1
--      Position2   Integer2
--      ...
--      Number_Of_Characters
--      Position1   Character1
--      Position2   Character2
--      ...
--      Number_Of_Booleans
--      Position1   Boolean1
--      Position2   Boolean2
--      ...
--      Number_Of_Floats
--      Position1   Float1
--      Position2   Float2
--      ...
procedure Load_Comps(Filename : in    string) is
    Infile : FILE_TYPE;
    Temp   : system.address;
    Flag   : integer;
begin
    open(Infile, in_file, Filename);

    -- Initialize RODB Integer Component
    for I in 1..RCDT.INT_SIZE loop
        Temp := Shmaddr + system.offset((I-1)*4);
        RCDT.AINT(Temp, 0);
    end loop;
    Load_Ints(Infile);

    -- Initialize RODB Character Component
    for I in 1..RCDT.CHAR_SIZE loop

```

```

    Temp := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
    RCDT.ACHAR(Temp, 'X');
end loop;
Load_Chars(Infile);

-- Initialize RODB Boolean Component
for I in 1..RCDT.BOOL_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
    RCDT.ABOOL(Temp, true);
end loop;
Load_Bools(Infile);

-- Initialize RODB Float Component
for I in 1..RCDT.FLT_SIZE loop
    Temp := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
    RCDT.AFLT(Temp, 0.0);
end loop;
Load_Flts(Infile);

close(Infile);

Flag := RCDT.SEMSINIT(Semid);
if Flag = -1 then
    raise Sem_Exception;
end if;

exception
when name_error =>
    put_line("File cannot be opened.");
    put_line("Loading components fails!");
when data_error | end_error =>
    put_line("File format is incompatible.");
    put_line("Loading components fails!");
when Sem_Exception =>
    put_line("Semaphore cannot be initialized.");
    raise Sem_Exception;
when others =>
    put_line("Unknown exception.");
    put_line("Loading components fails!");
end Load_Comps;

-- Save RODB Components to a disk file
-- The structure of the disk file is as following:
--     Number_Of_Integers
--     Position1 Integer1
--     Position2 Integer2
--     ...
--     Number_Of_Characters
--     Position1 Character1
--     Position2 Character2
--     ...
--     Number_Of_Booleans
--     Position1 Boolean1
--     Position2 Boolean2
--     ...
--     Number_Of_Floats
--     Position1 Float1
--     Position2 Float2
--     ...
procedure Save_Comps(Filename : in string) is

```

```

Outfile : FILE_TYPE;
begin
  if Filename /= "" then
    create(Outfile, out_file, Filename,
           form=>"world=>read, owner=>read_write");

    Save_Ints(Outfile);
    Save_Chars(Outfile);
    Save_Bools(Outfile);
    Save_Flts(Outfile);
    close(Outfile);
  else
    Save_Ints(TEXT_IO.standard_output);
    Save_Chars(TEXT_IO.standard_output);
    Save_Bools(TEXT_IO.standard_output);
    Save_Flts(TEXT_IO.standard_output);
  end if;
exception
  when constraint_error =>
    put_line("RODB Components data collapsed.");
    put_line("Saving components fails!");
  when others =>
    put_line("Unknown exception.");
    put_line("Saving components fails!");
end Save_Comps;

```

```

-- Shutdown RODB Components
procedure Shutdown_Comps is
  Flag : integer;
begin
  Flag := RCDT.SHMDT(Shmaddr);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SHMCTL(Shmid, 0, system.null_address);
  if Flag = -1 then
    raise Shm_Exception;
  end if;
  Flag := RCDT.SEMSRMV(Semid);
  if Flag = -1 then
    raise Sem_Exception;
  end if;
end Shutdown_Comps;

```

```
pragma page;
```

```

-- Load all the integers from a disk file to RODB Integer Component
procedure Load_Ints(Infile : in FILE_TYPE) is
  Length : Integer;
  Temp_Pos : integer;
  Temp_Int : integer;
  Temp_Addr : system.address;
begin
  INT_IO.get(Infile, Length);
  skip_line(Infile);
  for I in 1..Length loop
    INT_IO.get(Infile, Temp_Pos);
    INT_IO.get(Infile, Temp_Int);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.INT_SIZE-1) then
      raise Shm_Outrange;
    end if;
  end loop;
end Load_Ints;

```

```

        end if;
        Temp_Addr := Shmaddr + system.offset(Temp_Pos*4);
        RCDT.AINT(Temp_Addr, Temp_Int);
    end loop;
end Load_Ints;

-- Load all the charaters from a disk file to RODB Character Component
procedure Load_Chars(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Char   : character;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        get(Infile, Temp_Char);
        get(Infile, Temp_Char);           -- Skip a space
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.CHAR_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET+Temp_Pos);
        RCDT.ACHAR(Temp_Addr, Temp_Char);
    end loop;
end Load_Chars;

-- Load all the booleans from a disk file to RODB Boolean Component
procedure Load_Bools(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Bool   : boolean;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
        BOOL_IO.get(Infile, Temp_Bool);
        skip_line(Infile);
        if (Temp_Pos < 0) or (Temp_Pos > RCDT.BOOL_SIZE-1) then
            raise Shm_Outrange;
        end if;
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET+Temp_Pos);
        RCDT.ABOOL(Temp_Addr, Temp_Bool);
    end loop;
end Load_Bools;

-- Load all the floats from a disk file to RODB Float Component
procedure Load_Flts(Infile : in FILE_TYPE) is
    Length      : Integer;
    Temp_Pos    : integer;
    Temp_Flt    : float;
    Temp_Addr   : system.address;
begin
    INT_IO.get(Infile, Length);
    skip_line(Infile);
    for I in 1..Length loop
        INT_IO.get(Infile, Temp_Pos);
    end loop;
end Load_Flts;

```



```

    FLT_IO.get(Infile, Temp_Flt);
    skip_line(Infile);
    if (Temp_Pos < 0) or (Temp_Pos > RCDT.FLT_SIZE-1) then
        raise Shm_Outrange;
    end if;
    Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET+Temp_Pos*4);
    RCDT.AFLT(Temp_Addr, Temp_Flt);
end loop;
end Load_Flts;

```

```
pragma page;
```

```
-- Save all the integers from RODB Integer Component to a disk file
procedure Save_Ints(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
```

```

begin
    put(Outfile, "Number Of Integers is: ");
    Int_IO.put(Outfile, RCDT.INT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.INT_SIZE loop
        put(Outfile, "Integer number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset((I-1)*4);
        Int_IO.put(Outfile, RCDT.FINT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Ints;

```

```
-- Save all the characters from RODB Character Component to a disk file
procedure Save_Chars(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
```

```

begin
    put(Outfile, "Number Of Characters is: ");
    Int_IO.put(Outfile, RCDT.CHAR_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.CHAR_SIZE loop
        put(Outfile, "Character number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.CHAR_OFFSET + I-1);
        put(Outfile, RCDT.FCHAR(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Chars;

```

```
-- Save all the booleans from RODB Boolean Component to a disk file
procedure Save_Bools(Outfile : in FILE_TYPE) is
    Temp_Addr : system.address;
```

```

begin
    put(Outfile, "Number Of Booleans is: ");
    Int_IO.put(Outfile, RCDT.BOOL_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.BOOL_SIZE loop
        put(Outfile, "Boolean number ");
        Int_IO.put(Outfile, I-1, width => 5);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.BOOL_OFFSET + I-1);
        Bool_IO.put(Outfile, RCDT.FBOOL(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Bools;

```

```

    end loop;
end Save_Bools;

-- Save all the floats from RODB Float Component to a disk file
procedure Save_Flts(Outfile : in      FILE_TYPE) is
    Temp_Addr : system.address;
begin
    put(Outfile, "Number Of Floats is ");
    Int_IO.put(Outfile, RCDT.FLT_SIZE);
    new_line(Outfile);
    for I in 1..RCDT.FLT_SIZE loop
        put(Outfile, "Float number ");
        Int_IO.put(Outfile, I-1);
        put(Outfile, ":");
        Temp_Addr := Shmaddr + system.offset(RCDT.FLT_OFFSET + (I-1)*4);
        Flt_IO.put(Outfile, RCDT.FFLT(Temp_Addr));
        new_line(Outfile);
    end loop;
end Save_Flts;

pragma page;

begin
    Shmid := RCDT.SHMGET(RCDT.SHMKEY, RCDT.SHM_SIZE, 1023);
    if Shmid = -1 then
        raise Shm_Exception;
    end if;
    Shmaddr := RCDT.SHMAT(Shmid, system.null_address, 0);
    -- if Shmaddr = system.null_address then
    --     raise Shm_Exception;
    -- end if;
    Semid := RCDT.SEMGET(RCDT.SEMKEY, 3, 1023);
    if Semid = -1 then
        raise Sem_Exception;
    end if;
end Rodb_Component;

```

```

rrr      oooo      oooo      ttttt
  r      o  o      o  o      t
        o  o      o  o      t
        o  o      o  o      t
        o  o      o  o      t
        oooo      oooo      tt

```

```

rrr      eeee      aaaa      ddd d      b      eeee      ggg g      cccc
  r      e   e      a      d  dd      b      e   e      g   gg      c   c
        eeeee      aaaaa      d  d      b      eeeee      g   g      c   c
        e   e      a   a      d  d      b      e   e      g   gg      c   c
        e   e      a   aa      d  dd      b      e   e      g   gg      c   c
        eeee      aaaa a      ddd d      b bbb      eeee      ggg g      cccc
                                   g   g      ..
                                   gggg      ..

```

Job: readbeg.c  
Date: Tue Apr 14 22:37:57 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf RREAD_START          = { 0,  1, 0};
struct sembuf RWAIT_NO_WRITE_LOCK = { 1,  0, 0};
struct sembuf RWAIT_NO_WRITE_DESIRE = { 2,  0, 0};

int readbeg(semid)
int semid;
{
    struct sembuf one_of_n_readers[3]; /* Three semaphore operations */
    int    flag;
    void    perror();

    /* Perform three semaphore operations */
    one_of_n_readers[0] = RWAIT_NO_WRITE_LOCK; /* Wait for no more writer */
    one_of_n_readers[1] = RWAIT_NO_WRITE_DESIRE; /* Wait for no more writer */
    one_of_n_readers[2] = RREAD_START; /* Prevent writer in */
    flag = semop(semid, one_of_n_readers, 3); /* Lock the critical section */
    if (flag == -1) {
        perror("readbeg fails: ");
    }
    return flag;
}

```

```

      t
      t
      t
rrr   oooo   oooo   ttttt
  r   o   o   o   o   t
      o   o   o   o   t
      o   o   o   o   t
      o   o   o   o   t
      oooo   oooo   tt

```

```

      d
      d
      d
      d
rrr   eeee   aaaa   ddd d   eeee   n nnn   ddd d   cccc
  r   e   e   a   d   dd   e   e   nn   n   d   dd   c   c
      e   e   a   d   d   e   e   n   n   d   d   c   c
      e   e   a   d   d   e   e   n   n   d   dd   ..   c
      eeee   aaaa a   ddd d   eeee   n   n   ddd d   ..   cccc

```

Job: readend.c  
Date: Tue Apr 14 22:37:58 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on the semaphore */
struct sembuf RREAD_END      = { 0, -1, 0};

int readend(semid)
int semid;
{
    int flag;
    void perror();

    /* Perform the semaphore operation */
    flag = semop(semid, &RREAD_END, 1);
    if (flag == -1) {
        perror("readend fails: ");
    }
    return flag;
}

```

[illegible]

```
Job:  writebeg.c
Date:  Tue Apr 14 22:38:20 1992
```

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf WWAIT_NO_READERS = { 0, 0, 0};
struct sembuf WREAD_START      = { 0, 1, 0};
struct sembuf WWRITE_LOCK      = { 1, 1, 0};
struct sembuf WWRITE_DESIRE     = { 2, 1, 0};
struct sembuf WIN_PROGRESS_WRITE = { 2, -1, 0};

int writebeg(semid)
    int semid;
{
    struct sembuf sole_writer[4]; /* Four semaphore operations */
    int    flag;
    void    perror();

    /* Make write request by doing a semaphore operation */
    flag = semop(semid, &WWRITE_DESIRE, 1);
    if (flag == -1) {
        perror("Write-Request in writebeg fails: ");
        return flag;
    }

    /* Perform four semaphore operations */
    sole_writer[0] = WWAIT_NO_READERS; /* Wait for no more readers */
    sole_writer[1] = WWRITE_LOCK;      /* preventing succeeding readers */
    sole_writer[2] = WREAD_START;      /* preventing succeeding writers */
    sole_writer[3] = WIN_PROGRESS_WRITE; /* Cancel the write-request */
    flag = semop(semid, sole_writer, 4); /* Lock the critical section */
    if (flag == -1) {
        perror("Write_Start in writebeg fails: ");
    }
    pause();
    return flag;
}

```



```

rrr      oooo      oooo      t
r        o  o  o  o  ttttt
         o  o  o  o  t
         o  o  o  o  t
         o  o  o  o  t
         o  o  o  o  t
         oooo      oooo      tt

```

```

          i          t          d
          ii         ttttt         d
          i          t          d
          i          t          d
          i          t          d
          i          t  t          d
          iii        tt          ddd d
w  r rrr          eeee      eeee      n nnn      ddd d
w w rr      r    e   e      e   e      nn  n      d  dd
w w r          eeeee      eeeee      n    n      d  d
w w r          e   e      e   e      n    n      d  dd
w w r          eeee      eeee      n    n      ddd d
w ww r

```

Job: writeend.c  
Date: Tue Apr 14 22:38:35 1992

```

#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/sem.h>

/* The operations on semaphores */
struct sembuf WREAD_END      = { 0, -1, 0};
struct sembuf WWRITE_UNLOCK = { 1, -1, 0};

int writeend(semid)
    int semid;
{
    struct sembuf sole_writer[2]; /* Two semaphore operations */
    int    flag;
    void    perror();

    /* Perform the semaphore operation */
    sole_writer[0] = WREAD_END;
    sole_writer[1] = WWRITE_UNLOCK;
    flag = semop(semid, sole_writer, 2);
    if (flag == -1) {
        perror("writeend fails: ");
    }
    return flag;
}

```

```

      t
      t
rrr   oooo   oooo   ttttt
  r   o   o   o   o   t
      o   o   o   o   t
      o   o   o   o   t
      o   o   o   o   t
      oooo   oooo   tt

```

```

                                     i
                                     i
                                     t
                                     t
                                     ttttt
:sss   eeee   m m mm   ssss   ii   n nnn   ii   t
  s     e     mm m m   s  s     i   nn  n   i   t
:s     eeeee   m m m   ss     i   n  n   i   t
  ss    e     m m m   ss     i   n  n   i   t
  s     e     m m m   s  s     i   n  n   i   t
:sss   eeee   m m m   ssss   iii  n  n   iii  tt  ..

```

Job: semsinit.c  
 Date: Tue Apr 14 22:38:55 1992

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semsinit(semid)
    int semid;
{
    short initarray[3];
    int    flag;
    void    perror();

    initarray[0] = initarray[1] = initarray[2] = 0;
    flag = semctl(semid, 3, SETALL, initarray);
    if (flag == -1) {
        perror("semsinit fails: ");
    }
    return(flag);
}
```

```

rrr      oooo      oooo      t
      r  o  o  o  o  ttttt
      o  o  o  o  t
      o  o  o  o  t
      o  o  o  o  t
      oooo      oooo      tt

```

```

:sss      eeee      m m mm      ssss      r rrr      m m mm      v      v      cccc
      s  e      e      mm m m      s      s      rr      r      mm m m      v      v      c
:s      eeeeeee      m m m      ss      r      m m m      v      v      c
      ss      e      m m m      ss      r      m m m      v      v      c
      s  e      e      m m m      s      s      r      m m m      v      v      c
:sss      eeee      m m m      ssss      r      m m m      v      v      c

```

Job: semsrmv.c  
 Date: Tue Apr 14 22:39:02 1992

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semrmv(semid)
    int semid;
{
    int    flag;
    void    perror();

    flag = semctl(semid, 3, IPC_RMID, 0);
    if (flag == -1) {
        perror("semrmv fails: ");
    }
    return(flag);
}
```

```
rrr      r          oooo           ddd d   d b bbb cccc         oooo       m m mm    p pp     ..  
        r    o   o   o   o   o   dd  d   b bb  b  c      c   o   o   mm  m   m  pp    p  .  
      o   o   o   o   o   d   d   b   b   b   c      c   o   o   m   m   m  p    p  
      o   o   o   o   d   d   b   b   b   c      c   o   o   m   m   m  pp   p  
      oooo            ddd d   b bbb cccc         oooo       m   m   m  p  ppp  ..
```

```
Job:  rodbcomp.dat
Date:  Tue Apr 14 22:41:09 1992
```

```
10
0 100
1 200
2 300
3 400
4 500
5 600
6 700
7 800
8 900
9 1000
10
0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
8 I
9 J
10
0 false
1 false
2 false
3 false
4 false
5 false
6 false
7 false
8 false
9 false
10
0 100.0
1 200.0
2 300.0
3 400.0
4 500.0
5 600.0
6 700.0
7 800.0
8 900.0
9 1000.0
```





---

Copies of this publication have been deposited with the Texas State Library in compliance with the State Depository Law.

---